VA FILEMAN
SQL INTERFACE (SQLI)
SITE MANUAL

Patch DI*21.0*38

October 1997

Department of Veterans Affairs
V*IST***A** Software Development
Open**V***IST***A** Product Line

Table of Contents

SQLI Web Site: Companion to this Manual

The SQLI web site is an **online companion** to this manual. It contains up-to-date information about SQLI, including additional material not published in this manual.

Areas on the SQLI web site include:

- Latest News
- Documentation, including any updates
- FAQs (compiled from ongoing site experiences)
- Troubleshooting Tips (compiled from ongoing installation experiences)
- Demonstrations of SQLI projects
- Case Studies (updated beyond this manual's publication date)
- Vendor/Product Listings (added to as companion products to SQLI become available)

Check the SQLI web site **before** you install SQLI to get acquainted with the latest installation issues and solutions that the SQLI team is able to provide.

Also check the SQLI web site periodically **after** you start using SQLI, to get the latest information, ideas, troubleshooting tips, and other SQLI news.

The SQLI web site is located at:

http://www.vista.med.va.gov/softserv/infrastr.uct/sqli

Orientation

Typographic Conventions

At some places in this manual, you are shown a simulation of your interaction with your computer. In order to distinguish computer-supplied prompts from your responses, responses are in bold type. Like this:

```
COMPUTER'S PROMPT: USER'S RESPONSE
```

VA FileMan Information

Additional information about VA FileMan is available on the VA FileMan home page:

http://www.vista.med.va.gov/softserv/infrastr.uct/fileman/

For information about VA FileMan, consult its documentation set. VA FileMan manuals of particular interest to M-to-SQL vendors are:

- *VA FileMan V. 21.0 Programmer Manual*
- *VA FileMan V. 21.0 User Manual*

These manuals contain detailed information on VA FileMan, including its data dictionary structures, data format, field types, and API calls. They are available in both hardcopy and Adobe Acrobat PDF formats. Manuals in PDF format are available from the VA FileMan home page.

Additional SQLI Information

Additional information about SQLI is available on the SQLI home page:

http://www.vista.med.va.gov/softserv/infrastr.uct/sqli/

On that home page, an additional SQLI manual is available, targeted for M-to-SQL vendors:

- *VA FileMan SQLI Vendor Manual*

SQL Training

This manual does not attempt to explain relational database concepts, SQL queries, or how to access ODBC data sources. For this information, you should consult the documentation provided with the relational database products you are using. You may want to purchase training in these areas as well.

This manual does provide guidance about how VA FileMan files and fields may be projected through SQL and ODBC.

VA FileMan and SQL Terminology

The following table lists the equivalent terminology between VA FileMan and SQL.

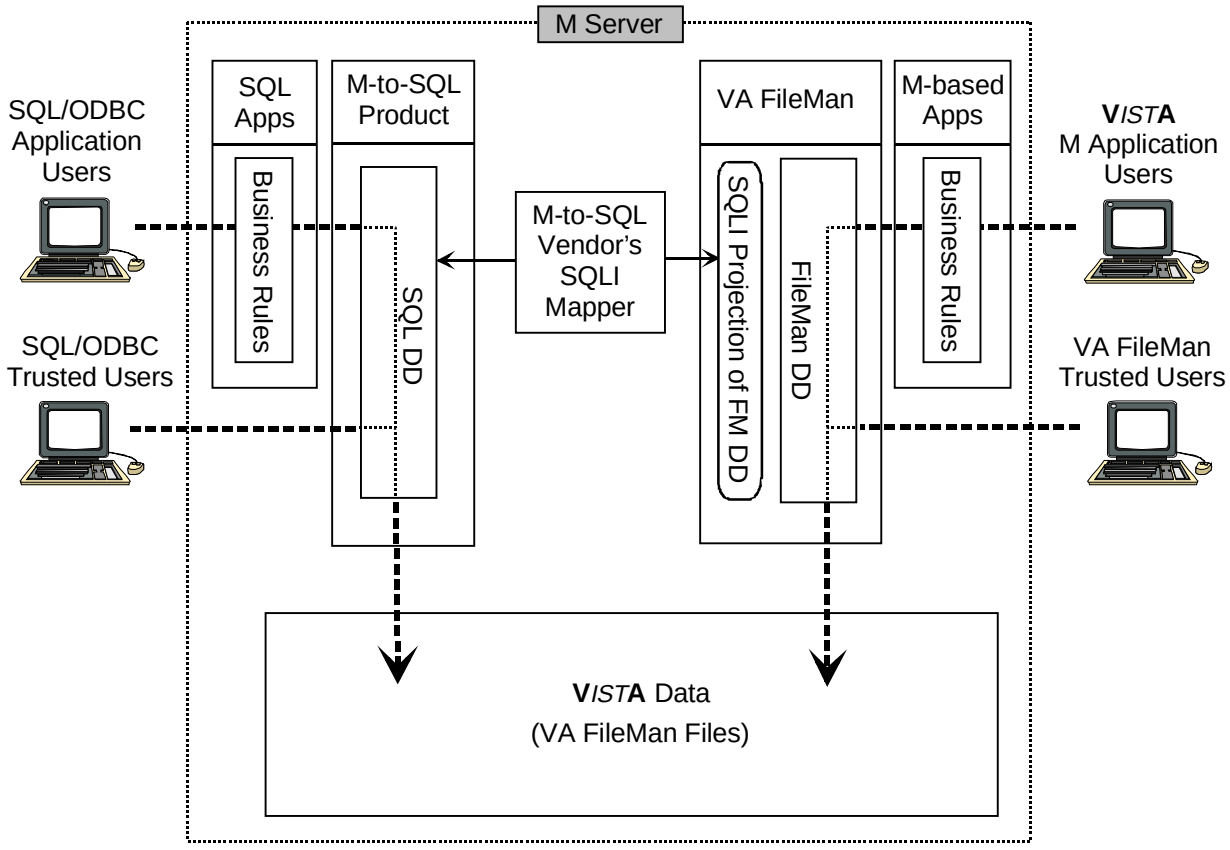
VA FileMan	SQL
n/a	Schema
File or Multiple	Table
Field	Column
Field Type	Domain
Record	Row

1. Introduction

From an operational point of view, VA FileMan is a hierarchical database, because it supports multiples (nested tables). However, the nature of VA FileMan's underlying file and data dictionary structures allow its files and multiples to be viewed as relational tables as well.

M-to-SQL vendors have provided relational access to VA FileMan data in the past by scanning VA FileMan's internal data dictionary structures, and interpreting the information found there to provide a relational view of the underlying data.

SQLI insulates the M-to-SQL vendors from direct access to VA FileMan's internal data dictionaries by projecting all information needed for a relational view of VA FileMan in a supported manner. Instead of accessing internal data dictionary structures, M-to-SQL vendors need only scan the SQLI's projection to build their relational view of VA FileMan data.



Why Map VA FileMan to SQL?

SQL (Structured Query Language) is the predominant language and set of facilities for working with relational tables.

The reason to access VA FileMan data through SQL is to take advantage of features provided both by SQL and by Commercial Off-the-Shelf (COTS) software. For example, if a vendor's M-to-SQL product can act as an ODBC data source, this provides direct access to VA FileMan data for ODBC-enabled Windows software!

Some ways you can work with VA FileMan data using an M-to-SQL product include:

- Query VA FileMan data through SQL.
- Manipulate VA FileMan data in Open Database Connectivity (ODBC) aware applications. This is possible if the M-to-SQL product can act as an ODBC data source. In this case, VA FileMan data can be accessed and manipulated by ODBC-compatible applications (Access, Excel, ReportSmith, Crystal Reports, etc.)
- Make VA FileMan data accessible over an Intranet or the Internet using an ODBC-aware web server.

What Other Software is Required to View VA FileMan Data Through SQL?

To enable SQL access to VA FileMan data at your site, you need to purchase an M-to-SQL product (software that can view structured M globals as relational tables through SQL).

The M-to-SQL vendor must also provide SQLI mapper software that map their SQL data dictionaries, based on SQLI's projection, to directly access VA FileMan data.

Mapping VA FileMan to SQL without SQLI

Several commercial vendors have already implemented SQL interfaces to the VA FileMan database, by mapping directly to VA FileMan's internal data dictionary structures. Despite their success, you should consider using an M-to-SQL product that can construct its mapping to the supported relational view of VA FileMan data structures provided by SQLI.

Advantages of Mapping to VA FileMan through SQLI

SQLI publishes all of the information M-to-SQL vendors need to access VA FileMan data through SQL. It places a layer between M-to-SQL vendors and VA FileMan's data dictionary, projecting the format for SQL access to VA FileMan files in a uniform manner. This approach results in a number of advantages, compared to vendors mapping directly against VA FileMan's internal data dictionary:

SQLI Feature	Advantage
SQLI projects VA FileMan files as tables and VA FileMan field types as SQL data types, functions, and domains.	Insulates M-to-SQL vendors from directly accessing VA FileMan's data dictionary structures, which are subject to change.
SQLI provides standard interpretations of data structures such as pointer fields, variable pointer fields, word processing fields, multiples, and internal entry numbers. It also provides a standard treatment of primary keys.	Enables standard approaches to writing queries across all VA sites.
SQLI publishes standard SQLI identifiers for each VA FileMan file and field.	Enables one site's SQL queries to work across all VA sites.
SQLI provides a standard layer in VA FileMan for SQL access features.	The presence of SQLI lays the groundwork for deeper integration of SQL access with VA FileMan.
SQLI provides code for many of the data retrieval tasks inherent in accessing VA FileMan data through SQL.	Should save vendor work, and encourage uniform approaches to retrieving VA FileMan data among vendors.

Note to Developers

As a developer, you may want to work with VA FileMan data relationally, using embedded SQL commands. You might then wonder if SQLI and its APIs would help you in doing this.

SQLI doesn't itself provide an API for directly accessing VA FileMan data. Nor is it able to provide access to VA FileMan data relationally on its own. Instead, it provides a framework for M-to-SQL vendors to access VA FileMan's internal data dictionary information. M-to-SQL vendors are then able to provide relational access to VA FileMan data.

As a developer, to work with VA FileMan data relationally your application should make use of services provided by COTS M-to-SQL software. Your application can then use SQL statements to access VA FileMan data, either directly or in conjunction with ODBC-aware client applications. Using services provided by the COTS M-to-SQL product (and, optionally, ODBC-aware client software), your application can:

- Run SQL queries on VA FileMan data
- Implement business rules using SQL stored procedures
- Create and use database views
- Optionally incorporate business rules (as SQL stored procedures) in database views

2. Installing an M-to-SQL System Using SQLI

Components of an M-to-SQL System Using SQLI

Providing SQL access to your VA FileMan data requires using a number of software components, all working together. **V***IST***A**'s SQLI software product is one component of several needed for a complete system.

SQLI	Part of VA FileMan. Projects VA FileMan's internal data dictionary information in a format tailored for use by M-to-SQL vendors.
M-to-SQL Product	Must be purchased from a vendor. It should: • Provide SQL services • Be able to map its SQL data dictionaries so that SQL tables can reference data stored in M globals.
SQLI Mapper	Provided by the vendor of the M-to-SQL product. This utility reads the information published by SQLI to map the M-to-SQL product's SQL data dictionaries to reference VA FileMan data.
ODBC Drivers	(Optional) If the M-to-SQL software can act as an ODBC data source, the vendor can provide ODBC workstation drivers.

Steps to Provide Relational Access to VA FileMan

A. Determine Desired Type(s) of Access to VA FileMan Data

B. Choose and Purchase M-to-SQL Vendor's Software

C. Anticipate Training Needs

D. Install and Configure M-to-SQL System

E. Do the First SQLI Projection and Mapping

A. Determine Desired Type(s) of Access to VA FileMan Data

Two general types of access to VA FileMan data have been demonstrated through M-to-SQL products:

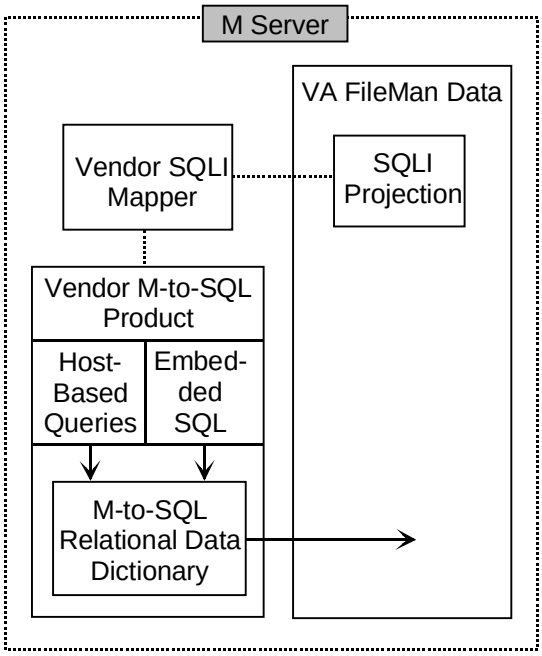
- SQL host-based queries on VA FileMan data
- ODBC client access to VA FileMan data

The type of access you need affects your purchasing decision for M-to-SQL products, and your infrastructure needs for end-users.

SQL Host-Based Queries on VA FileMan Data

A host-based query is simply an SQL query done through an M-to-SQL vendor's native software interface. To enable host-based queries, you only need to set up an M-to-SQL product to access VA FileMan data. The M-to-SQL product should provide some type of interface to allow SQL queries. Typically this is a character-based SQL interface, allowing queries to be entered and results returned on a character-based terminal.

Sample System Configuration (Host-Based Queries)



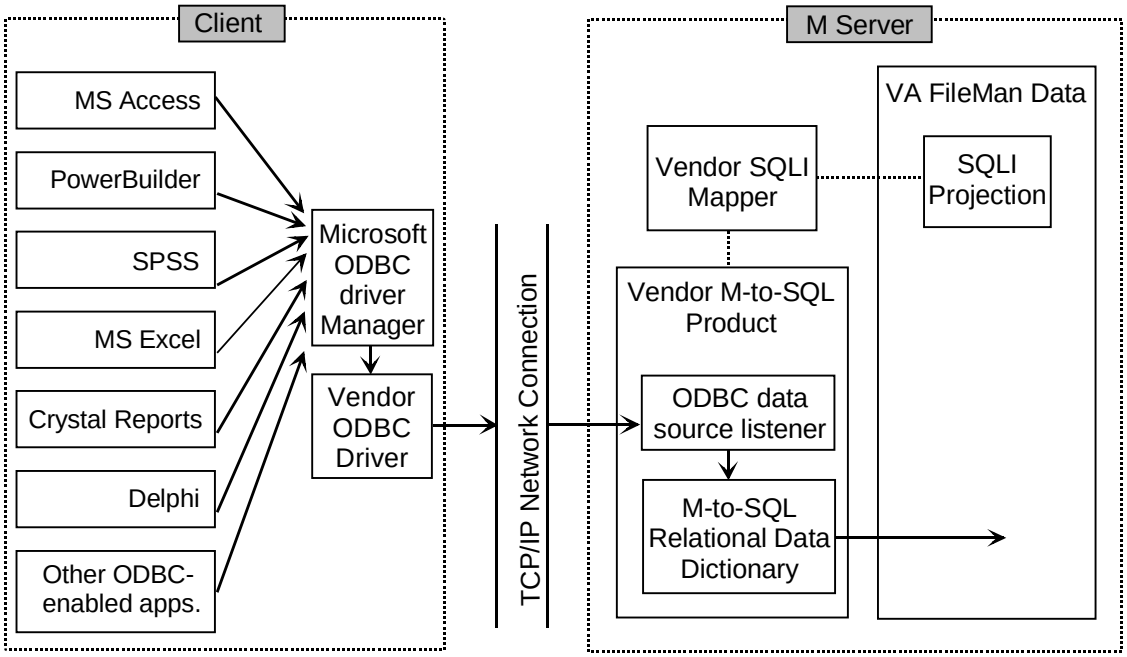
ODBC Client Access to VA FileMan Data

ODBC (Open Database Connectivity) is the most common way to connect MS-Windows clients to host-based database systems. ODBC insulates MS-Windows client programs from the details of connecting to a specific database; instead, the client connects to an ODBC driver. The ODBC driver in turn connects to a database that can act as an ODBC data source. The driver handles the details of accessing the database in question.

If you purchase M-to-SQL software that can act as an ODBC data source, then MS-Windows clients can access your VA FileMan data through ODBC. For this level of support, in addition to the requirements for host-based queries:

- The M-to-SQL database software must also be able to act as an ODBC data source.
- The M-to-SQL vendor must provide ODBC drivers for client workstations to connect to the M-to-SQL ODBC data source.
- A TCP/IP network must connect the client workstations to the M-to-SQL database software.
- The client software must be ODBC-enabled.

Sample System Configuration (ODBC)



B. Choose and Purchase M-to-SQL Vendor's Software

The purpose of using SQLI at your site is to take advantage of a vendor-provided M-to-SQL product.

A key factor in choosing a vendor's M-to-SQL product is the availability and pricing of an SQLI mapper utility from the vendor. This utility reads the information published by SQLI to map the M-to-SQL product's SQL data dictionaries to reference VA FileMan data. The availability and pricing of an SQLI mapper utility is within the purview of the M-to-SQL vendor.

Performance and cost-per-user are factors to consider for most software; M-to-SQL software is no exception.

Other factors beyond performance and cost to consider when choosing a vendor's software are:

- ODBC Support
- SQLI Mapper Implementation

ODBC Support

Some M-to-SQL products can act as ODBC data sources. As described in the previous section, to enable ODBC access to VA FileMan data, the M-to-SQL product must:

- Act as an ODBC data source.
- Provide ODBC client drivers for end-user workstations to access the ODBC data source.

SQLI Mapper Implementation

M-to-SQL vendors map their SQL data dictionaries to the SQLI projection. There is a considerable amount of freedom for vendors in how they use the information in SQLI to project tables and columns, and data values. Because the mapping process is up to the vendor, the level of implementation of SQLI's projection of VA FileMan can vary among vendors. Also, the SQL implementations themselves vary from vendor to vendor, which can affect how some types of data (e.g., word processing fields) are presented.

Also, some features that are available through host-based queries may not be available through ODBC.

Some of the issues to consider when evaluating a vendor's SQLI mapper are:

- Namespacing. Has the vendor received a formal namespace from the VHA DBA? If not, does the vendor's namespace conflict with a VA namespace?
- Programming SAC (Standards and Conventions). Does the vendor's code conform to VA's Programming SAC?
- Pointer fields. Will the end-user see resolved external values through host-based queries? Through ODBC?
- Pointer fields. Is the internal pointer value available to perform joins with through host-based queries? Through ODBC?
- Set of codes fields. Will the end-user see resolved external values through host-based queries? Through ODBC?
- Word processing fields. Are they presented as columns using a memo-like data type at the "parent" table level (i.e., the entire contents of the word processing field is available as a column value), or as standalone tables where each line of text is an individual row that must be joined with the "parent" table? If a memo-like data type is used, how is truncation handled if word processing field data exceeds the maximum size of the memo field?
- Foreign keys. Is foreign key or similar functionality implemented that makes joins based on pointer fields and multiples easier?
- Update/Insert/Delete. The first version of SQLI does not provide specific support for writes to the database. Does the vendor's M-to-SQL implementation disable update/insert/delete access to tables mapped from SQLI? If not, how has the vendor implemented these operations?
- DBA control. Is a full suite of DBA features available to restrict access to SQL tables (and hence, VA FileMan data)?

C. Anticipate Training Needs

DBA Training

The security mechanism for VA FileMan files accessed through M-to-SQL software is the M-to-SQL software's Database Administrator (DBA) mechanism. Your DBA should be fully trained in the security features of your SQL software, before you implement SQL access to VA FileMan data.

First, you should designate who will be the DBA(s) for your M-to-SQL product. A DBA is an SQL user with all SQL privileges, including the ability to create other SQL users, and to grant and revoke access.

Then, the designated DBA(s) should familiarize themselves and be trained in all aspects of security in the M-to-SQL database software, including:

- Setting up SQL users
- Granting and revoking user, schema, table, column, and view privileges
- Creating database views
- Listing and auditing all users and privileges

End-User Training

Your end-users will need to be trained in whatever software they'll use to access VA FileMan data. Skills needed by your end-users (depending on what products they will use to access data) may include:

- Relational database concepts (tables, columns, primary keys, joins, etc.)
- Writing SQL queries
- Accessing ODBC data sources from Windows software (Excel, Access, etc.)
- Understanding how VA FileMan fields are projected through SQL and ODBC

This manual does not attempt to explain relational database concepts, SQL queries, or how to access ODBC data sources. If training in these areas is needed, your users should consult the documentation provided with the products they are using and/or pursue training in these areas as well.

This manual does provide information about how VA FileMan files and fields may be projected through SQL and ODBC (see the "SQLI for End-Users" chapter).

D. Install and Configure M-to-SQL System

Required VISTA Software

- Kernel V. 8.0, fully patched - needed to install KIDS (Kernel Installation and Distribution System) distribution of SQLI
- VA FileMan V. 21.0, fully patched
- The SQLI patch (DI*21*38), installed

Skills Required

- Configure M system parameters such as stack buffer, line buffer, and global structure size (these parameters may vary among M vendors).
- Manage globals, including global placement, protection, translation, and journaling characteristics.
- Create and schedule options.
- Install, configure, and operate third-party SQL products.
- Perform SQL queries.
- Use DBA (database administrator) functions in the third-party SQL products to manage access to SQL tables.

Disk Space Needed

The SQLI projection of a VA FileMan database containing all current national **VISTA** software packages uses approximately 30 megabytes of disk space. Use of locally developed files could add additional space to the mapping, so a total of up to 40 megabytes of disk space usage can be anticipated.

The M-to-SQL database software will also use disk space, both for the SQL implementation itself, and also for the SQL data dictionaries generated by the vendor's SQLI mapping software. The space used will vary among vendors; consult their documentation for more information on the disk space usage.

Set Up M-to-SQL System

Installing and configuring M-to-SQL database software is an essential part of the method of mapping VA FileMan data relationally.

If No M-to-SQL Service Exists

1. Prepare a volume set and UCI having global access to ^DIC, ^DD and all application globals. When choosing the UCI and volume set for the M-to-SQL software, remember that the SQL database software may create large temporary tables when performing queries, which could consume significant amounts of disk space.
2. Set up the UCI, stack buffer, line buffer and global node size according to SQL vendor recommendations.
3. Verify that the routine namespaces and global usage of the SQL vendor do not conflict with VA or local usage.
4. Load, initialize and check vendor M-to-SQL software as recommended by the vendor.

If M-to-SQL Service Exists

Sites with existing M-to-SQL services may have some or all of their VA FileMan files already mapped in their SQL product's data dictionaries. If these mappings in the SQL product data dictionaries are not removed, there is a high potential for conflicting table names if the SQL mapping is done to a new schema, or for inappropriate merging if the SQL mapping is done to the same schema. It's best to start with a clean slate.

1. From the vendor SQL data dictionary, delete **table definitions only** that were previously mapped directly from ^DD. Make sure that doing this does not delete the associated **data**. Follow vendor recommendations.
2. Archive and delete all vendor software and files associated with mapping directly from ^DD. Follow vendor recommendations.
3. Verify that the routine namespaces and global usage of the SQL vendor do not conflict with VA or local usage.

E. Do the First SQLI Projection and Mapping

The first SQLI projection of VA FileMan files will be the hardest, because the projection process uncovers most of the anomalies in your system's data dictionary structures. When it encounters an anomaly that it hasn't been designed handle gracefully, the SQLI projection process will likely crash. Each time an anomaly is encountered, you need to fix the anomaly and then re-run the SQLI projection until it is able to complete.

1. Check the SQLI web site for the latest information and troubleshooting tips.
2. Confirm that VA FileMan patch DI*21*38 (the SQLI patch) is loaded.
3. You should have already designated who will be the DBA (database administrator) on the SQL system. Make sure the DBA has changed the default DBA password. The DBA's account assigns security to SQL tables, create schemas, grant access to tables, etc. through SQL commands. It is essential that this account be secure.
4. It is **not** necessary to remove users or tasks from the system when running the SQLI projection, or to inhibit logons.
5. (Optional) Run VA FileMan's Check/Fix DD Structure [DI DDUCHK] option for all files on your system (for a range, start with the lowest numbered file, and go to the highest numbered file on your system). Answer Yes when it asks whether it should "Remove Erroneous Nodes". This will correct some of the data dictionary anomalies in advance that might otherwise cause the SQLI mapping process to abort. For more information on this option, see the *VA FileMan User Manual*.
6. Install and configure the vendor supplied SQLI mapper software as recommended by vendor. This would typically be an add-on to the already-installed vendor M-to-SQL product.
7. Using vendor tools, perform any pre-mapping initialization the SQLI mapper vendor specifies, including loading the SQLI_KEY_WORD file with any keywords specific to your M-to-SQL vendor.
8. Using the SQLI "Run the Regenerate SQLI Projection" option [DMSQ PROJECT], create the SQLI projection. This compiles the VA FileMan data dictionary into the SQLI files. It takes several hours to compile a large application set, depending on the number of files to project, the speed of your system, and the load on your system.

   Make sure no changes to the VA FileMan data dictionaries are made while the SQLI projection is running (no patches or packages should be installed, and no programmers should directly modify VA FileMan data dictionary definitions.)

```
Select SQLI (VA FileMan) Option: REgenerate SQLI Projection <RET>
This process takes several hours.  Want to Continue? NO// YES

     Running this job on your terminal (HOME device) will tie up
     your terminal for the several hours it takes to run, but you
     will see the job's status as it's running.

     Queuing will send it to the background for processing.  The
     status will be apparent from the printed output (if there's an
     error, it's text will be printed).  TaskMan/Kernel tools can also
     be used to determine whether the job ran to completion or not.

     Don't send this directly to a printer (without queuing) unless
     you are prepared to tie up your terminal AND the printer for
     the duration of the process.

DEVICE: HOME// <RET>

Table 4676                    Time elapsed: 00:50:12 (HH:MM:SS)
Columns of 4676               Time elapsed: 03:11:59 (HH:MM:SS)
Foreign key 5258              Time elapsed: 03:25:25 (HH:MM:SS)
Index 4676                    Time elapsed: 04:31:18 (HH:MM:SS)
```

You can use the Find Out SQLI Status option to determine the current status of the currently running or last completed SQLI projection.

If the mapping encounters an anomaly in the data dictionary structure (which it will probably do several times during your first mapping) it will abort with an error. You need to determine and fix the data dictionary anomaly that aborted the projection process, and then re-run the SQLI projection from the beginning. See the Troubleshooting Errors that Abort SQLI section in the SQLI System Management chapter for a complete troubleshooting procedure for data dictionary anomalies.
After a full run of the SQLI projection completes, check out errors generated by the process (see "Errors Generated During the SQLI Projection" below).

9. Now that the SQLI projection has been built, you're ready to map the vendor M-to-SQL product's data dictionaries based on the SQLI projection.

   Run the vendor-provided SQLI mapper as recommended by the vendor. This mapper uses the information published by SQLI to set up the M-to-SQL vendor's SQL data dictionaries. It will probably take several hours to compile a large application set, depending on the number of files to map, the speed of your system, and the load on your system.

10. The person who is acting as the SQL DBA may now set up SQL users and grant them the appropriate privileges to access only the tables that they need.

After the First Mapping

1. Update existing queries. If you had M-to-SQL services in place prior to installing SQLI, and you ran SQL queries on VA FileMan data, you should debug your existing VA FileMan SQL queries. Make sure that they still operate correctly in the new SQLI environment. Primarily you will need to correct any schema, table and column names in your queries that may have changed.
2. Check disk space (optional). You may want to monitor the disk space used both by SQLI (for its projection of VA FileMan data dictionaries) and by your M-to-SQL product (for its mapping of VA FileMan data structures).

Provide Access to SQLI Options

The following options are intended for IRM use to manage SQLI:

Option Text	Option Name	Locked?
Regenerate SQLI Projection	DMSQ PROJECT	XUPROGMODE
Print Errors from Last Projection	DMSQ PRINT ERRORS	
Purge SQLI Data	DMSQ PURGE	XUPROGMODE

The following options are intended for DBAs and other interested users:

Option Text	Option Name	Locked?
Table Statistics Reports	DMSQ TS MENU	
Site Statistics Reports	DMSQ PS MENU	
Suggest Table Groupings	DMSQ SUGGEST TABLE GROUPINGS	

SQLI Implementation Notes

- **.001 number fields**. The optional .001 number field for a file, if defined, represents the ien of entries. Such fields are not projected as columns by SQLI. You can access this value using the TABLE_ID column (the ien column), which SQLI does project for all tables.
- **Asterisked files.** Any files or subfiles whose names start with an asterisk are not projected in SQLI. Note: Adding an asterisk to the beginning of a field name is a VA Programming SAC convention to mark the field as obsolete.
- **Dangling pointers**. It is possible that a VA FileMan field may contain a pointer to a file not actually present at a given site. If so, the field is projected as a normal pointer field would be, but without the corresponding output format that permits navigation along a pointer chain to resolve the external value of the pointer. Such fields are flagged in the SQLI_ERROR_LOG during SQLI generation as "Pointer to Absent Files". Foreign keys for such fields are not constructed.
- **Field attributes not projected**. Along with number, the following field attributes are projected by SQLI: Label, field length, type, specifier, global subscript location, pointer, multiple-valued, and the first line of the field's description. Other field attributes, including output transforms and pointer screens, are not projected. See the *VA FileMan V. 21.0 Programmer Manual*, Global File Structure chapter, for more information about field attributes.
- **File Attributes not projected**. Only file name and number are projected. Other file attributes, such as Special Lookup and Screens, are not. See the *VA FileMan V. 21.0 Programmer Manual*, Global File Structure chapter, for more information about file attributes.
- **Files not in ^DIC**. Only files with entries in ^DIC (the dictionary of files) are projected. This means only VA FileMan-compatible files are projected.
- **Internal VA FileMan tables not projected**. Certain tables used by VA FileMan internally (numbered below two) are not projected. Errors are logged during SQLI projection in the SQLI_ERROR_LOG. VA FileMan DD numbers in this category include .001, .1, .12, .15, .21, .3, 1.001 and 1.01.
- **Multiline computed fields**. Values are not returned for multiline computed fields, since DBS calls can't retrieve multiline computed fields. An example of a multiline computed field is a backward extended pointer reference.
- **Non-regular cross-references**. Only regular VA FileMan cross-references are projected. VA FileMan Trigger, KWIC (Key Word in Context), MUMPS, Mnemonic, Soundex, and Bulletin type indexes are absent from SQLI. Cross-references are only projected for possible optimizations by M-to-SQL vendors.

- **Output transforms**. Output transforms are not projected. If formatting needs to be applied, it can be applied at the SQL vendor column level. For more elaborate output transforms that may call routines for processing, the logic will need to be reproduced in the context of the query. Depending on your M-to-SQL product's capability, the external value of a field (after the output transform is applied) could be returned by a user-defined function that invokes the VA FileMan $$EXTERNAL^DILF API call.
- **Variable pointers**. Variable pointers are projected as text only. Their text value is resolved, but presented as text.

3. SQLI System Management

Re-Projecting SQLI when File Structures Change

When a patch or installation changes the structure of VA FileMan files, the SQL data dictionaries of the M-to-SQL product that accesses those files **must** be updated. Otherwise, users accessing data through the SQL data dictionaries may access inaccurate or erroneous data, or may error out.

To update your M-to-SQL product's SQL data dictionaries:

1. Disable SQL Access while Remapping

   When running either the SQLI projection or the vendor's SQLI mapping utilities, disable access to your SQL users. To shut down ODBC access, shut down the ODBC data source listener process on the SQL server. To prevent host-based query access, use whatever utility the SQL vendor provides.

2. Perform any pre-mapping initialization the vendor's SQLI mapper vendor specifies, including loading the SQLI_KEY_WORD file with any keywords specific to your M-to-SQL vendor. This should be done each time SQLI is projected, because there could be new keywords (for example, a new user-defined SQL function could impact the keyword list.)

3. Run the Regenerate SQLI Projection Option [DMSQ PROJECT]

   This option purges and then rebuilds the SQLI projection for all VA FileMan files on the system. Errors during the compilation are logged in the SQLI_ERROR_LOG file.

```
Select SQLI (VA FileMan) Option: REgenerate SQLI Projection <RET>
This process takes several hours.  Want to Continue? NO// YES

     Running this job on your terminal (HOME device) will tie up
     your terminal for the several hours it takes to run, but you
     will see the job's status as it's running.

     Queuing will send it to the background for processing.  The
     status will be apparent from the printed output (if there's an
     error, it's text will be printed).  TaskMan/Kernel tools can also
     be used to determine whether the job ran to completion or not.

     Don't send this directly to a printer (without queuing) unless
     you are prepared to tie up your terminal AND the printer for
     the duration of the process.
```

```
DEVICE: HOME// <RET>

Table 4676                    Time elapsed: 00:50:12 (HH:MM:SS)
Columns of 4676               Time elapsed: 03:11:59 (HH:MM:SS)
Foreign key 5258              Time elapsed: 03:25:25 (HH:MM:SS)
Index 4676                    Time elapsed: 04:31:18 (HH:MM:SS)
```

4. Remap M-to-SQL

   Once you have regenerated the SQLI projection, use the vendor-provided SQLI mapping utility to remap the M-to-SQL data dictionaries to the new SQLI projection.

Determining the Current Status of the Projection

Use the Find Out SQLI Status option to determine the current status of the currently running or last completed SQLI projection.

Scheduling the SQLI Projection on a Regular Basis

You may want to schedule a re-projection of SQLI and a full vendor SQLI remapping to run automatically, perhaps once per month. This is to make sure all VA FileMan data dictionary changes get reflected in the SQLI projection.

You can use Task Manager to schedule the Regenerate SQLI Projection option [DMSQ PROJECT] on a regular basis. This regenerates the SQLI projection.

See the section below, Set Up SQLI Projection and Vendor Mapping as One Task, for how you can modify this option so that it can also invoke the vendor's utilities. Then, a complete remapping (SQLI' projection plus the vendor's SQLI mapping) can be performed on a regular basis.

SQLI Projection and Vendor Mapping as One Task

The Regenerate SQLI Projection [DMSQ PROJECT] option does not automatically perform the vendor side of the mapping. It only creates the SQLI projection. You may want to schedule a task that does both sides (SQLI and vendor) to completely regenerate the SQLI system. To run all of the programs necessary to completely refresh the SQL view of VA FileMan, you can make a local version of the DMSQ PROJECT option. This local option can invoke the necessary vendor utilities in its Entry and Exit actions, with SQLI's projection being run as the option's main routine. This local option can then be scheduled through Task Manager.

Scheduling the Vendor Mapping

If the vendor who supplies your SQLI mapper has made that utility Kernel-compatible, you can run both the SQLI Projection and the vendor SQLI mapping as a single task.

To do this, copy SQLI's DMSQ PROJECT option to a local option (perhaps DMSQZ PROJECT). Then, modify the Entry Action and Exit Action fields of your local DMSQZ PROJECT option as follows:

```
ENTRY ACTION: ;vendor's keyword call could go here
EXIT ACTION: ;vendor's mapper call could go here
```

Any vendor utility that should run before the SQLI projection should be called in the Entry Action of your local DMSQZ PROJECT option. Updating the SQLI KEY WORD file is a typical task to run before the SQLI projection.

Any vendor utility that should run after the SQLI projection should be called in the Exit Action of your local DMSQZ PROJECT option. The vendor SQLI mapping is a typical task to run after the SQLI projection. If the SQLI projection errors out, the Exit Action will not be executed.

Kernel Compatibility

For a vendor's utilities to be Kernel compatible, they should conform to the VA Programming SAC. This includes the setting and killing of variables, the ways that devices are used, and not interfering with Kernel error trapping.

For TaskMan compatibility, the vendor SQLI mapper must be able to run non-interactively. Also, if the vendor's utilities need to write to host files, OpenVMS sites must be running Task Manager in a DCL context. For more information on running Task Manager in a DCL context, see the *Kernel V. 8.0 Systems Manual*.

Troubleshooting Errors that Abort SQLI

If a very unusual non-standard data dictionary structure is present in a VA FileMan file's data dictionary, it's possible that either the SQLI regeneration or the vendor remapping process could encounter a hard error. If a hard error occurs, you should try the following steps:

1. Determine the file and field that was being processed at the time the hard error occurred. Use the Find Out SQLI Status [DMSQ DIAGNOSTICS] option to determine on what file and field the projection process stopped. You can also look at the error trap and at the last entries in the SQLI files to help figure out in what file and field the projection process stopped.
2. Try to determine if there is anything unusual in the VA FileMan data dictionary for that file and/or field.
3. If there is no obvious problem in the file's data dictionary, run VA FileMan's Check/Fix DD Structure option for the file in question, to check and fix any irregularities this option finds in the data dictionary. For more information on this option, see the *VA FileMan User Manual.*
4. Try running the RUNONE^DMSQ direct mode utility for the file in question. This utility tries to remap a single file. If it succeeds, you have probably fixed the data dictionary anomaly. If it fails, you probably need to continue to look for the problem. This utility is for diagnostic testing purposes only, and is not supported as a means of rebuilding a file's SQLI projection.
5. If the RUNONE^DMSQ direct mode utility fails, check the SQLI web site. A list of known issues that cause trouble for SQLI will be posted there. Its URL is:

   http://www.vista.med.va.gov/softserv/infrastr.uct/sqli

6. If the problem still cannot be determined, obtain support (through NOIS or other current support process) if you can't resolve the problem in the preceding steps.
7. As a last resort, if it is a local file or subfile, you could mark the file name obsolete with an asterisk, e.g. *FILE. The SQLI projection ignores files whose names begin with "*".

Once you have fixed the data dictionary anomaly that caused the SQLI projection to crash, run the SQLI projection again.

Errors Messages from SQLI Projections

To list the non-fatal error messages generated during the initial or subsequent projections of VA FileMan data dictionaries by SQLI, use the Print Errors from Last Projection option [DMSQ PRINT ERRORS]. You can also examine the contents of the SQLI_ERROR_LOG file directly through VA FileMan.

Errors are to be expected during SQLI projection. When SQLI encounters a non-standard data dictionary structure, it may not project the field or file. For example, some of VA FileMan's internal files (those numbered less than two) show up as errors in the SQLI error listing.

Typically, the result of encountering an error condition is that the file or field in question will not be projected. So, in most cases you do not need to review the error list unless there's a particular file or field you need that did not get projected.

SQLI Error Messages

The following is a complete list of error messages that can be reported when generating the SQLI projection.

```
      COLUMN:  CAN'T GET FIELD ELEMENTS
      COLUMN:  DECIMAL DEFAULT IS NEGATIVE
      COLUMN:  FIELD TYPE NOT KNOWN TO SQLI
      COLUMN:  INSERT OF COLUMN ELEMENT FAILED
      COLUMN:  INSERT OF COLUMN RECORD FAILED
      COLUMN:  INVALID FIELD LABEL
      COLUMN:  NO ASSOCIATED TABLE
      COLUMN:  NO CORRESPONDING TABLE ELEMENT
      COLUMN:  NULL FIELD TYPE (DOMAIN)
   DATA TYPE:  INSERT OF DATA TYPE RECORD FAILED
      DOMAIN:  INSERT OF DOMAIN RECORD FAILED
       FIELD:  CALL TO RETRIEVE ATTRIBUTES FAILED
        FILE:  CAN'T BUILD SQL NAME
        FILE:  INSERT OF TABLE FAILED
        FILE:  NO DESCRIPTION
        FILE:  NO GLOBAL ROOT
        FILE:  NO NAME
        FILE:  NOT FILEMAN COMPATIBLE
        FILE:  NULL DESCRIPTION
        FILE:  OBSOLETE
        FILE:  SUBFILE WITHOUT PARENT
FOREIGN KEY:  ANCESTOR FOREIGN KEY COLUMN INSERT FAILED
FOREIGN KEY:  ANCESTOR FOREIGN KEY INSERT FAILED
FOREIGN KEY:  COLUMN ELEMENT INSERT FAILED
FOREIGN KEY:  NO ANCESTOR PRIMARY KEY
FOREIGN KEY:  NO ASSOCIATED PRIMARY KEY
FOREIGN KEY:  NO POINTED-TO COLUMN AT LEVEL
FOREIGN KEY:  NO POINTED-TO FILE IN SPECIFIER
```

```
      FOREIGN KEY:  NO PRIMARY KEY TABLE ELEMENT
      FOREIGN KEY:  NO TABLE FOR POINTED-TO FILE
      FOREIGN KEY:  TABLE ELEMENT INSERT FAILED
INDEX PRIMARY KEY:  CAN'T GET COLUMN'S TABLE ELEMENT
INDEX PRIMARY KEY:  CAN'T GET DATA FOR MASTER TABLE
INDEX PRIMARY KEY:  COLUMN ELEMENT INSERT FAILED
INDEX PRIMARY KEY:  COLUMN INSERT FAILED
INDEX PRIMARY KEY:  MISSING COLUMN POINTER
INDEX PRIMARY KEY:  MISSING TABLE RECORD
INDEX PRIMARY KEY:  TABLE ELEMENT INSERT FAILED
INDEX PRIMARY KEY:  TABLE MISSING COLUMN POINTER
            INDEX:  COLUMN ELEMENT INSERT FAILED
            INDEX:  COLUMN INSERT FAILED
            INDEX:  IRREGULAR FORMAT
            INDEX:  MISSING DATA DICTIONARY DATA
            INDEX:  NO ASSOCIATED COLUMN RECORD
            INDEX:  PRIMARY KEY ELEMENT INSERT FAILED
            INDEX:  PRIMARY KEY INSERT FAILED
            INDEX:  TABLE DOMAIN INSERT FAILED
            INDEX:  TABLE INSERT FAILED
       KEY FORMAT:  LONG_CHARACTER INSERT FAILED
             ONEF:  NO PARENT STRUCTURE
   OUTPUT FORMAT:  INSERT OF POINTER OUTPUT FORMAT FAILED
   OUTPUT FORMAT:  INSERT OF SET-OF-CODES OUTPUT FORMAT FAILED
   OUTPUT FORMAT:  INSERT OF VARIABLE POINTER OUTPUT FORMAT FAILED
     PRIMARY KEY:  CAN'T GET TABLE DATA
     PRIMARY KEY:  CAN'T GET TABLE'S FILE #
     PRIMARY KEY:  DOMAIN INSERT FAILED
     PRIMARY KEY:  TABLE ELEMENT INSERT FAILED
          SCHEMA:  RECORD INSERT FAILED
           STATS:  KEY COUNT INSERT FAILED
           STATS:  RECORD INSERT FAILED
         SUBFILE:  BAD UP-LINK TO PARENT
```

Errors Generated During SQL Access

Besides errors that occur while projecting SQLI, it is possible that errors may occur when an M-to-SQL product mapped to VA FileMan data attempts to access the VA FileMan data. Possible causes are:

- Conditions in the M database that the M-to-SQL vendor did not anticipate.
- Conditions in VA FileMan data that the SQLI mapper vendor did not anticipate.

When this type of error occurs, you should first look at the data being accessed to see if there is anything unusual about that data. If nothing appears unusual, you should contact the M-to-SQL vendor and SQLI mapper vendor for assistance troubleshooting the error.

DBA Security: Managing Access to VA FileMan Data

SQL includes ANSI-standard security mechanisms for SQL security. You should use these mechanisms, implemented through your M-to-SQL software's DBA functions, to provide security for VA FileMan data projected through SQL.

Selectively Grant Access to VA FileMan Data

Recommendations for administering security on VA FileMan files projected through SQLI to an SQL system are as follows:

1. By default, VA FileMan files projected as tables are assigned to a schema named "SQLI". Don't grant blanket access to all files in the SQLI schema (in effect, every table) to end-users.
2. Decide what sets of tables, rows and columns are needed for each set of users.
3. Define **views** of the database for your users that reflect specific sets of tables, rows, and columns that are useful to specific user groups.
4. Grant access both by view and by table, as appropriate, to your SQL end-users and/or user groups.
5. Regularly audit your list of users and user groups along with their access to tables, rows, columns, and views.

Don't Allow Updates to VA FileMan File Data from SQL

This release of SQLI (patch DI*21*38) does not provide any support for vendors to implement commands that write to VA FileMan files (INSERT, UPDATE, DELETE). This lack of support **doesn't** prevent a vendor from implementing these commands, however.

One significant problem with direct writes from SQL is that VA business rules are typically stored in application code, not in the file data dictionaries. So many business rules would not be executed when updating a VA FileMan file from SQL.

So, except for situations in which you are sure all business rules will be executed, the DBA should make sure that no users are granted the following SQL privileges:

- DELETE
- INSERT
- UPDATE
- REFERENCES (data dictionary edit privileges)

Ensuring the Accuracy of the Mapping

Data must be VA FileMan-Compatible

Mapping SQL data dictionaries to the SQLI projection of VA FileMan data dictionaries assumes that the stored data, including indexes, is in a format that matches its VA FileMan data dictionary definition. If this is not the case (as can happen when data is hard-set into VA FileMan data globals instead of through an API call), the column definitions in SQL data dictionaries will not be accurate.

Don't Map Manually

If you are using SQLI, don't map your SQL data dictionaries to VA FileMan data globals manually (without SQLI). SQLI takes a number of steps to ensure the integrity of the projection of the VA FileMan data dictionaries. While your M-to-SQL software will most likely have an option to map globals directly to the SQL data dictionaries, make sure that all VA FileMan data is mapped from the SQLI projection.

How to Purge SQLI Data

If you decide to stop using SQLI, you may want to purge the SQLI data files and possibly your M-to-SQL vendor's SQL data dictionaries as well. To do this, you can use the Purge SLQI Data option [DMSQ PURGE].

```
Select SQLI (VA FileMan) Option: PUrge SQLI Data <RET>
Removes all records from SQLI files. Continue? NO// <YES>
```

Use your M-to-SQL vendor's comparable utility, if available, to purge all SQL data dictionaries mapped to VA FileMan file structures.

4. SQLI for End-Users

This chapter provides guidelines for end-users on how to interpret and work with VA FileMan data when it is viewed relationally, as projected by SQLI.

As an end-user of VA FileMan data accessed through SQL, you may or may not need to know any of the information in this chapter:

- If you are accessing data through an application such as a pre-designed web page that fully shields you from the underlying details of where pieces of data has come from, you probably don't need to know any of the information in this chapter.
- If, on the other hand, you are accessing the SQL table equivalents of VA FileMan data directly, either through SQL or through ODBC, and are dealing with tables, columns and rows, you probably do need to know the information provided in this chapter.

The main issues for end-users accessing the SQL table equivalents of VA FileMan data directly (as compared to accessing the same information through VA FileMan) are:

- Ways to access VA FileMan data through M-to-SQL
- Naming (tables and columns)
- Data format (columns)
- Joining Tables
- Business rules

Ways to Access VA FileMan Data Through M-to-SQL

Using an M-to-SQL product to provide relational access to VA FileMan data opens up a world of new possibilities when it comes to working with VA FileMan data. For example, you may be able to:

- **Directly Access the V*ISTA* Database from Windows Applications**

  If your site implements relational access such that ODBC access is enabled, you can use any ODBC-aware Windows application to directly access VA FileMan files. This means you can use your favorite Windows-based ODBC-aware applications to directly load, format and print records from the **V*ISTA*** database.

  To do this otherwise, you would need to use the VA FileMan Export Tool. You would export data to a host file in either character-delimited or width-delimited format, transfer the host file from the site computer to your personal computer, and then import the data into the PC-based application in the specified format.

- **Query the VISTA Database Using SQL**

  If your site implements M-to-SQL access to VA FileMan data, you should be able to query the **V*ISTA*** database using the native SQL interface of the M-to-SQL product. This may enable you to create different types of reports than you typically create using VA FileMan's report options.

- **Build Web Applications**

  Once VA FileMan data is accessible via ODBC, it is possible to build web interfaces to retrieve records in the **V*ISTA*** database and format and display those records as web pages. Building web applications is a task most likely undertaken by site developers.

Naming Issues

SQLI Naming Rules

SQLI provides standard naming for VA FileMan files and fields, as projected into SQL tables and columns. SQLI ensures that the names used in the SQL projection of VA FileMan don't violate SQL's or ODBC's rules for element naming. The naming rules used by SQLI include:

- Illegal characters such as spaces are replaced with underscores.
- Names more than 30 characters long are compressed by abbreviating each word in the name until a length of 30 characters is reached.
- Names must start with a letter from A to z.
- Names may contain only the letters A through z, digits 0 through 9 and the underscore character "_".

Consistency of Naming Supports Query Sharing between Sites

By using consistent naming algorithms for files and fields, SQLI increases the likelihood that table and column names generated for national files and fields between VA sites are the same. This allows queries to be shared between sites using SQLI with minimal changes. Only under unusual circumstances will the naming algorithms produce a different field or file name between sites.

VA FileMan Files as Tables

When looking at a VA FileMan database through a relational database, as projected by SQLI, the following should be accessible as tables:

- VA FileMan files
- VA FileMan subfiles (each subfile is projected as a standalone table)
- (optional) VA FileMan word processing fields

You would expect VA FileMan files projected as tables in a relational system, and they are projected as such by SQLI.

However, VA FileMan subfiles are also projected as standalone tables. This may not be your ordinary way of thinking about data that is in subfiles. However, this "flattening" of VA FileMan subfiles into standalone tables is necessary to project VA FileMan data relationally. It also may provide you new ways to analyze data that has been stored in subfiles.

Table Naming

The picture below shows a partial list of table names for VA FileMan top-level files and subfiles, as retrieved by Microsoft Access through ODBC. Note that this client software lists table names appended to a schema name (in this case, SQLI):

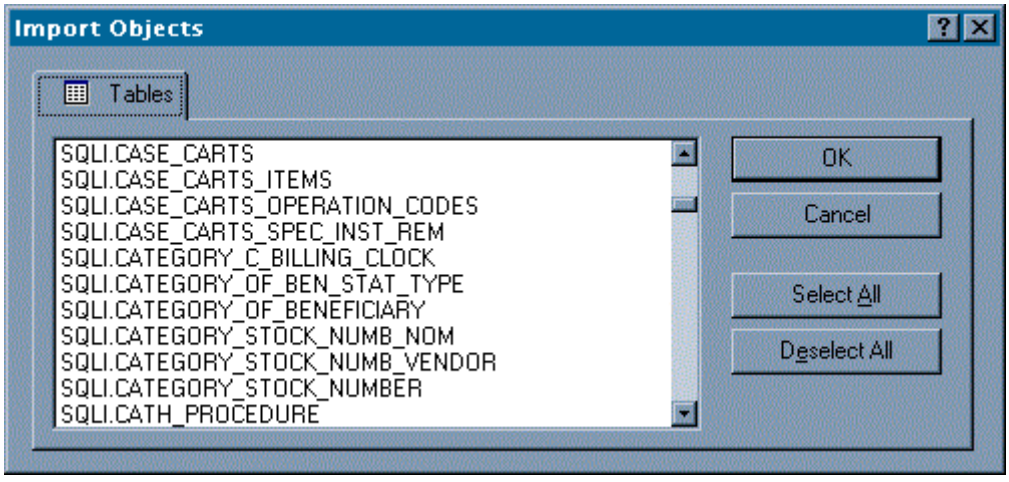


Table Names for Subfiles

A subfile (also called a multiple) is essentially a file-within-a-file: A hierarchical, record-specific multiple field. For example, a patient file might have an "Appointments" subfile. This file-within-a-file can contain one or more entries for the patient's appointments. Subfiles can themselves contain subfiles.

When viewed from a relational database, VA FileMan subfiles are "flattened" and presented as standalone tables. For more information about how subfiles are flattened, see "Flattening of Subfiles into Standalone Tables" later in this chapter.

Tables derived from subfiles use names based on top-level filename, all parent subfile names, and the subfile name itself, separated by "_"s.

For example, suppose that CASE_CARTS is a table projected for a top level file (CASE CARTS). The following are the table names projected for subfiles within the CASE CARTS file:

VA FileMan Subfile Name	Table Name Projected
ITEMS	CASE_CARTS_ITEMS
OPERATION CODES	CASE_CARTS_OPERATION_CODES
SPECIAL INSTRUCTIONS/REMARKS	CASE_CARTS_SPEC_INST_REM

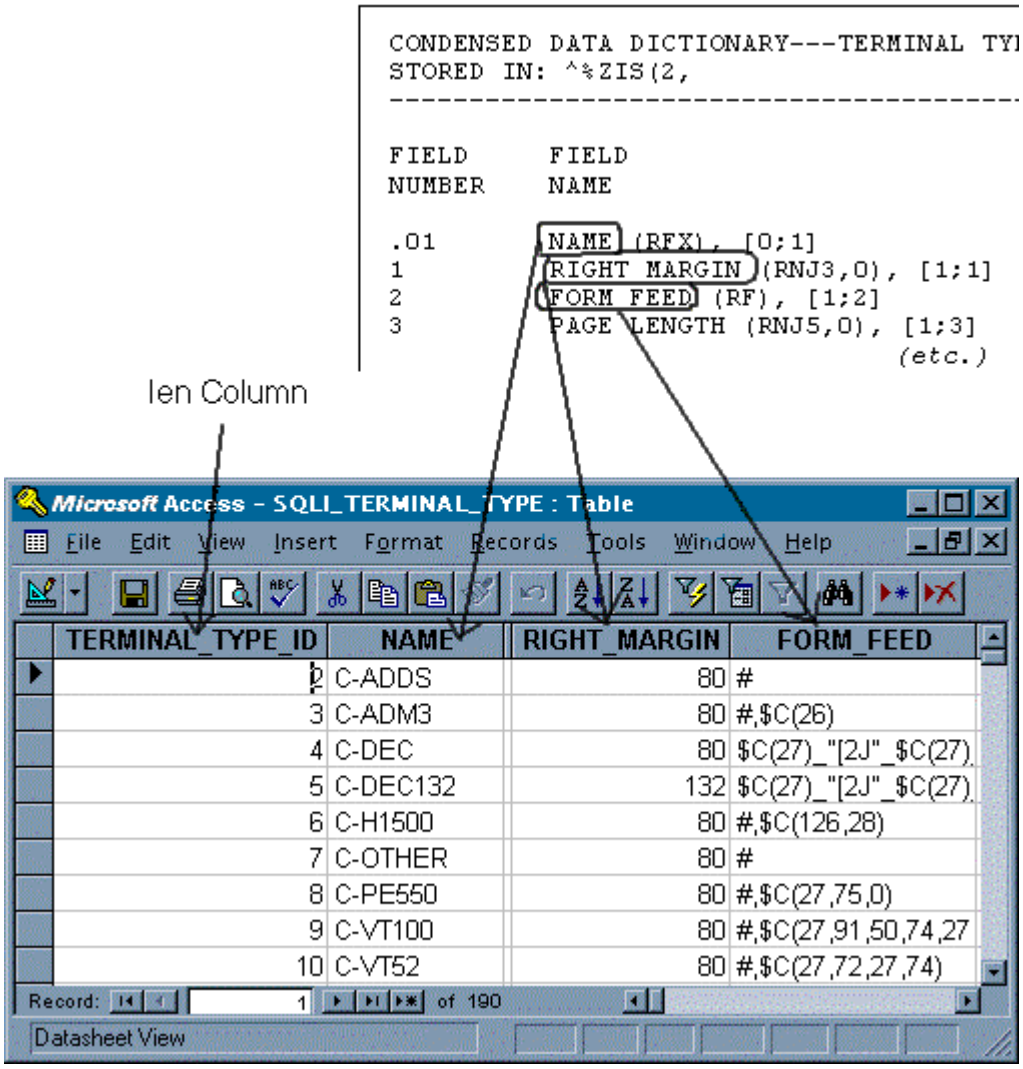
Schemas

SQLI projects all VA FileMan files as part of a single schema, "SQLI". Your relational database software may or may not make use of the schema designation of files.

Column Names for Fields

Column names are based on the field names. Adjustments in the projected name are only made if the original name violates SQL or ODBC naming rules.

In the example below, compare the projected column names for the SQLI_TERMINAL_TYPE table to the data dictionary field names for the fields in the TERMINAL TYPE file. Note the ien column as well, which corresponds to the iens of entries rather than to a data dictionary field.



DBA Reports: Table and Column Naming

A number of reports are available from SQLI options that provide statistical information about the SQLI projection of VA FileMan data.

In particular, the following reports provide a complete listing of the original VA FileMan naming of files and fields, compared to the naming by SQLI of the same files and fields:

- Table Name Listing (VA FileMan vs. SQLI)
- Field Listing by File (Brief)
- Field Listing by File (Full)

These reports are available from the main SQLI menu, under two options (Table Statistics Reports and Site Statistics Reports).

A full listing of available reports is as follows:

Table Statistics Reports **[DMSQ TS MENU]**

```
Field Listing by File (Brief)                          [DMSQ TS FIELDS BRIEF]
Field Listing by File (Full)                            [DMSQ TS FIELDS FULL]
List Subfile Links (Brief)                            [DMSQ TS SUBFILE BRIEF]
List Incoming Pointer/Subfile Links (Full)          [DMSQ TS PTR SUBFILE FULL]
List Pointer and Parent Links (Brief)               [DMSQ TS PTR PARENT BRIEF]
List Pointer and Parent Links (Full)                 [DMSQ TS PTR PARENT FULL]
Pointer Statistics by Individual Table                      [DMSQ TS PTR STATS]
Pointer Statistics (Summary)                      [DMSQ TS PTR STATS SUMMARY]
Table Name Listing (VA FileMan vs. SQLI)                      [DMSQ TS NAMES]
```

Site Statistics Reports **[DMSQ PS MENU]**

```
Table Total (Excluding Index Tables)                    [DMSQ PS TOTAL TABLES]
Column Total (All Tables)                              [DMSQ PS TOTAL COLUMNS]
Index Table Total                                      [DMSQ PS TOTAL INDEXES]
Table Element Totals, By Type                  [DMSQ PS TOTAL TABLE ELEMENTS]
Column Totals, by Table                            [DMSQ PS TOTAL TABLE COLS]
Column Totals, by Table (Ordered by # of Columns) [DMSQ PS TOTAL TABLE COLS A]
Columns in Regular Tables Total                  [DMSQ PS TOTAL COLUMNS REG]
Columns in Regular Tables, Excluding ID Columns   [DMSQ PS COLUMNS REG NOID]
Columns by Domain                                [DMSQ PS COLUMNS BY DOMAIN]
```

Data Format of Columns

VA FileMan field types roughly correspond to SQL data types. This section describes the specifics of how the data format of individual VA FileMan field types is affected when that data is viewed through SQL.

Base vs. External Data Values for Columns

Your M-to-SQL vendor may provide syntax for retrieving the base value of a column (as opposed to its external value).

When you do comparisons, such as a join that involves a pointer field, you may want the comparison to be done with the internal value of the field (ien) rather than the external value (text name), to ensure the join is done as VA FileMan would do it. A join on "SMITH, JOHN" might match multiple entries, whereas a join on "15553" would match a single entry.

When viewing data through ODBC, on the other hand, the base form of column values may be what is returned, rather than the external form. This may present difficulty if, through ODBC, you want the external form of a VA FileMan Set of Codes field.

The following table lists VA FileMan field types where base vs. external format may be an issue:

Field Type	Base	External
Pointer	Ien of pointed-to entry	Resolved pointer value
Variable Pointer	Ien of pointed-to entry;file global root	Resolved pointer value
Set of Codes	Internal Code	External Code

Your SQL vendor may provide syntax in their SQL for retrieving the base value of a column, or for resolving the external value of a column. This syntax is vendor specific; check with your SQL vendor for more information.

Internal Entry Number (Ien) Column

In VA FileMan, each entry in a file has an internal entry number (ien). When working with VA FileMan data from within VA FileMan, you usually don't see the ien of a record; you only see the field values of the record.

On the other hand, when you work with tables projected for VA FileMan files as projected by SQLI, each table provides an ien column to display the ien of each VA FileMan entry. The name of the ien column is the table name concatenated with "_ID". The ien column for each table is important, because it is used as the primary key for each record.

Tables projected for subfiles provide an ien column not only for the ien of subfile entries, but also ien columns for all file levels above the original subfile. Thus, with all the iens for all file levels available, each subfile record is uniquely identified.

Word Processing Fields

VA FileMan stores word processing fields in a manner very similar to how it stores multiples. Each line of text in a word processing field is stored as if it is an entry in a multiple field.

SQLI projects multiples to M-to-SQL vendors in two ways:

- As standalone tables (each line of text is one row in the table).
- As columns for vendors who support a HUGE_CHARACTER or MEMO data type.

Your M-to-SQL vendor may present word processing fields in an appropriate MEMO-like data type. The main problem with memo data types is that they usually come with a size constraint. VA FileMan word processing fields, on the other hand, are unlimited in size. So truncation could occur if a word processing field is encountered whose size exceeds the size defined as the maximum for a MEMO column.

If your M-to-SQL vendor does not provide an appropriate MEMO-like data type, word processing fields may instead be available as standalone tables. In this case, you will need to join each line (stored as individual rows in the word processing field table) with its parent table to access both at the same time.

Joining Tables

When viewed through VA FileMan, VA FileMan files can be joined in several ways. Pointer fields join an entry in one file with an entry in another file. Multiples are joined with their parent entries. Joins can be done on the fly in VA FileMan's Print File Entries module.

When viewed through SQL, you must perform joins between tables derived from VA FileMan files explicitly. Primary keys and ien columns are projected by SQLI to aid in this process.

Primary Keys

The primary key of a table identifies any row in the table uniquely. SQLI projects designated keys for VA FileMan-derived tables in a standard fashion:

- Tables for top-level VA FileMan files have a single-element key, which is the ien column for the table in question. The ien column is named as the table name appended with "_ID". Just as iens provide uniqueness for VA FileMan records, the ien column provides uniqueness for VA FileMan table rows.
- Tables for subfiles and word processing fields have multi-element primary keys, with one ien column present in the table for each parent file, plus one ien column for the ien of the subfile entry itself.

Primary keys are very useful when you need to join tables. Because they uniquely identify rows, they can be used to recreate the "join" relationships present in the original VA FileMan files, both for pointer fields and for multiples linking to their parents.

For an example of the primary key projected for a top-level file, take the table projected for the NEW PERSON file. A one-part primary key is projected for the NEW_PERSON table; it is the ien column for the table (NEW_PERSON_ID).

For an example of the primary key projected for a subfile, take the table projected for the DIVISION subfile in the NEW PERSON file. A two-part primary key is projected for the NEW_PERSON_DIVISION table, based on the following two columns (both present in the NEW_PERSON_DIVISION table):

```
NEW_PERSON_ID           (ien column of original parent entry)
NEW_PERSON_DIVISION_ID  (ien column of original subfile entry)
```

Recreating a Pointer Field Relationship between Tables

For columns derived from pointer fields, the base value of the column is the ien of the pointed-to entry. You can join the base value of the pointer field column to the ien column (primary key) of the pointed-to table. Use a where clause in your query to relate the pointer field column to the primary key of the pointed-to file:

```
WHERE POINTER_COLUMN = POINTED_TO_FILE_ID
```

For example, to join the PATIENT table to the NEW_PERSON table based on the PATIENT.PRIMARY_PHYSICIAN column (which is derived from a pointer field to the NEW PERSON file), a WHERE clause would be:

```
WHERE PATIENT.PRIMARY_PHYSICIAN=NEW_PERSON.NEW_PERSON_ID
```

Although you *could* join two tables based on the resolved text value of a pointer column, it's better to use the base value of the pointer column. This preserves referential integrity by assuring that identical resolved .01 field values wouldn't be joined in ways that don't reflect the original pointer relationship.

Losing Rows When a Pointer Field Column is Null

In the simple query example above, there is a problem when entries in the PATIENT table do not have a value in the column for PRIMARY_PHYSICIAN. When the tables are joined based on the PRIMARY_PHYSICIAN column, all rows in PATIENT where PRIMARY_PHYSICIAN is null are excluded from the join.

One way to get around this is to use an "outer join". Even if there is no matching row in a subfile's table, a row from the top-level file's table will be included in an outer join. Consult your SQL vendor's documentation to see if outer joins are supported, and, if so, what the outer join syntax is.

Another way to get around this is to use foreign keys (described in the next section).

Joining Tables Using Foreign Keys

A foreign key provides an explicit link between two SQL tables. The tables are in a sense "pre-joined". Using the foreign key in a query, you can access columns in the linked table, without having to perform a join yourself.

SQLI publishes foreign keys for M-to-SQL vendors to make use of, in the following standard situations:

Situation	Foreign Key(s) Provided
Column based on pointer field	In the table containing the pointer field column, one for the pointed-to file, named *pointer_field_name*_FK. The join is from the pointer field to the pointed-to table.
Table projected for subfile or word processing field	In the subfile or word processing field's table, one for each parent table, each named *parent_table*_PFK. Each join links the subfile to its original VA FileMan parent.

One advantage of using foreign keys rather than joins is that rows are not lost when the value of a join column is null. One way to think of this is that the "join" is being performed in the select clause of the query (which is where you can use foreign keys to include fields from other tables), rather than the where clause.

For example, a foreign key (i.e., NEW_PERSON_FK@NAME) can be used in the select clause to obtain the value of the column NAME from the NEW_PERSON table, rather than doing a join to NEW_PERSON in a where clause. A row is returned even if the NAME column of the corresponding row in the NEW_PERSON file is null.

Not all M-to-SQL vendors support the SQL concept of foreign keys. For those that do, the syntax to use foreign keys may be vendor-specific, so consult your M-to-SQL vendor's documentation for more information.

Flattening of Subfiles into Standalone Tables

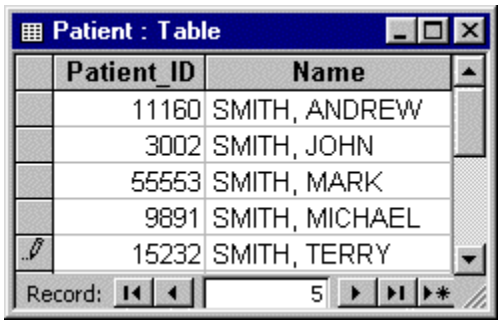
VA FileMan subfiles allows one record to store one or more repeating "subrecords". This form of data representation, while not supported in the relational model, is very useful for storing certain types of data, such as the set of one or more appointments for a patient.

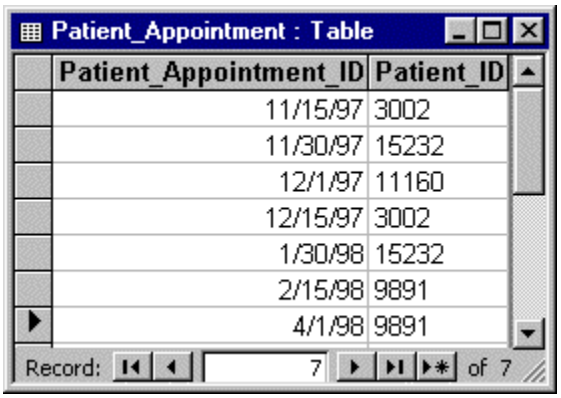
In VA FileMan, a listing of PATIENT records, including the APPOINTMENT subfile records, might look like:

```
PATIENT LIST                                    Oct  3,1997  14:58    PAGE 5
NAME                                  APPOINTMENT
------------------------------------------------------------------------------

SMITH, ANDREW                         12/01/1997
SMITH, JOHN                           11/15/1997
                                      12/15/1997
SMITH, MARK
SMITH, MICHAEL                        02/15/1998
                                      04/01/1998
SMITH, TERRY                          11/30/1997
                                      01/30/1998
```

In the relational model, repetitive information from VA FileMan multiples such as appointments are stored in standalone tables. In this example, one table holds the patient names while another holds the appointments. You can use the Patient_ID ien column to link each row in the Patient_Appointment table to the appropriate Patient table row:





In the example above, no record appears in the Patient_Appointment table for patient Mark Smith, who has no appointments.

Note The iens in the PATIENT_APPOINTMENT_ID column appear as dates. This is because the .001 field of the subfile is defined as a Date/Time type. This results in date/times being used as the internal entry number for appointment records.

Recreating the Relationship between a Subfile and its Parents

One of the challenges of working with VA FileMan data relationally is the splitting or flattening of subfiles that occurs in a relational view, and how to re-join the resulting subfile tables with their parent tables.

For a table projected for a subfile, a multi-part primary key is provided that provides the ien of each parent entry up to and including the top-level entry that originally enclosed the subfile. This means that the table for the subfile includes columns ending with "_ID", each one containing the ien of one of the subfile entry's parents.

You can use these primary key "_ID" columns in a join to recreate the relationship between a row in a subfile table and its parents.

For example, APPOINTMENT is a multiple in the PATIENT file. Therefore the table for the subfile, PATIENT_APPOINTMENT, has a two column primary key, with the following columns:

Primary Key Column	Function
PATIENT_APPOINTMENT_ID	ien of subfile entry
PATIENT_ID	ien of subfile's parent entry in PATIENT

To recreate the relationship between rows in the PATIENT_APPOINTMENT table with the "parent" PATIENT table, you could do:

```
SELECT * FROM PATIENT, PATIENT_APPOINTMENT
WHERE PATIENT.PATIENT_ID = PATIENT_APPOINTMENT.PATIENT_ID
```

Losing Parent Table Rows when Joining with Subfile Tables

When you join rows in a subfile table with rows in its parent table(s), you won't lose any rows from the subfile table under ordinary circumstances. This is because subfile rows always have corresponding rows in parent tables to join with.

However, rows in the parent tables are "lost" from the join if, in the original VA FileMan file, they did not have entries in their multiples. This is a problem if your query is trying to retrieve rows from parent tables regardless of whether or not they had subfile entries.

As with joins on pointer fields, one way to get around this is to use an *outer join*. Another way to get around this is to use foreign keys, if your M-to-SQL implementation supports them. In both cases, the syntax is vendor-specific, so consult your M-to-SQL vendor's documentation for more information.

Business Rules

Although some business rules for **V*IST*A** data are implemented in the data dictionaries of VA FileMan files, many business rules are not in the data dictionaries. Instead, these business rules are in application code. When you access VA FileMan data directly, either through VA FileMan's native interface (the ENTER OR EDIT FILE ENTRIES, SEARCH FILE ENTRIES and PRINT FILE ENTRIES options) or through SQL, you are bypassing any business rules not in the data dictionary.

Note also (as described in the Implementation Notes below) that output transforms are not projected by SQLI, so any business rules implemented in the output transform of a field are not used for data you access through SQL.

The same cautions that apply to accessing VA FileMan data directly through VA FileMan's native interface also apply to accessing that data through SQL: the data you are seeing may not be as the application developer intended it to be viewed, if the business rules are in application code. You must be careful to ensure that the data you are viewing is in the form you believe it to be in, and that you are aware of any business rules that would otherwise transform that data.

Insert/Update/Delete Commands

You may want to update VA FileMan files from SQL. Explicit support for vendors to implement Insert, Update, and Delete operations is not implemented in the first version of SQLI (patch DI*21*38).

A caution for implementing these types of access to VA FileMan data is that business rules are quite often not stored in VA FileMan data dictionaries. In particular, business rules for how to put data into the database quite often reside in application code, not in the data dictionary of a file. Bypassing **V*IST*A** application code and updating directly from SQL cannot execute business rules stored solely in application code, and can cause data corruption by circumventing those business rules.

5. SQLI Technical Information

SQLI File List

The following table lists the files set up when the SQLI DI*21*38 patch is installed:

Global Storage	File Number	File Name
^DMSQ("S",	1.521	SQLI_SCHEMA
^DMSQ("K",	1.52101	SQLI_KEY_WORD
^DMSQ("DT",	1.5211	SQLI_DATA_TYPE
^DMSQ("DM",	1.5212	SQLI_DOMAIN
^DMSQ("KF",	1.5213	SQLI_KEY_FORMAT
^DMSQ("OF",	1.5214	SQLI_OUTPUT_FORMAT
^DMSQ("T",	1.5215	SQLI_TABLE
^DMSQ("E",	1.5216	SQLI_TABLE_ELEMENT
^DMSQ("C",	1.5217	SQLI_COLUMN
^DMSQ("P",	1.5218	SQLI_PRIMARY_KEY
^DMSQ("F",	1.5219	SQLI_FOREIGN_KEY
^DMSQ("ET",	1.52191	SQLI_ERROR_TEXT
^DMSQ("EX",	1.52192	SQLI_ERROR_LOG

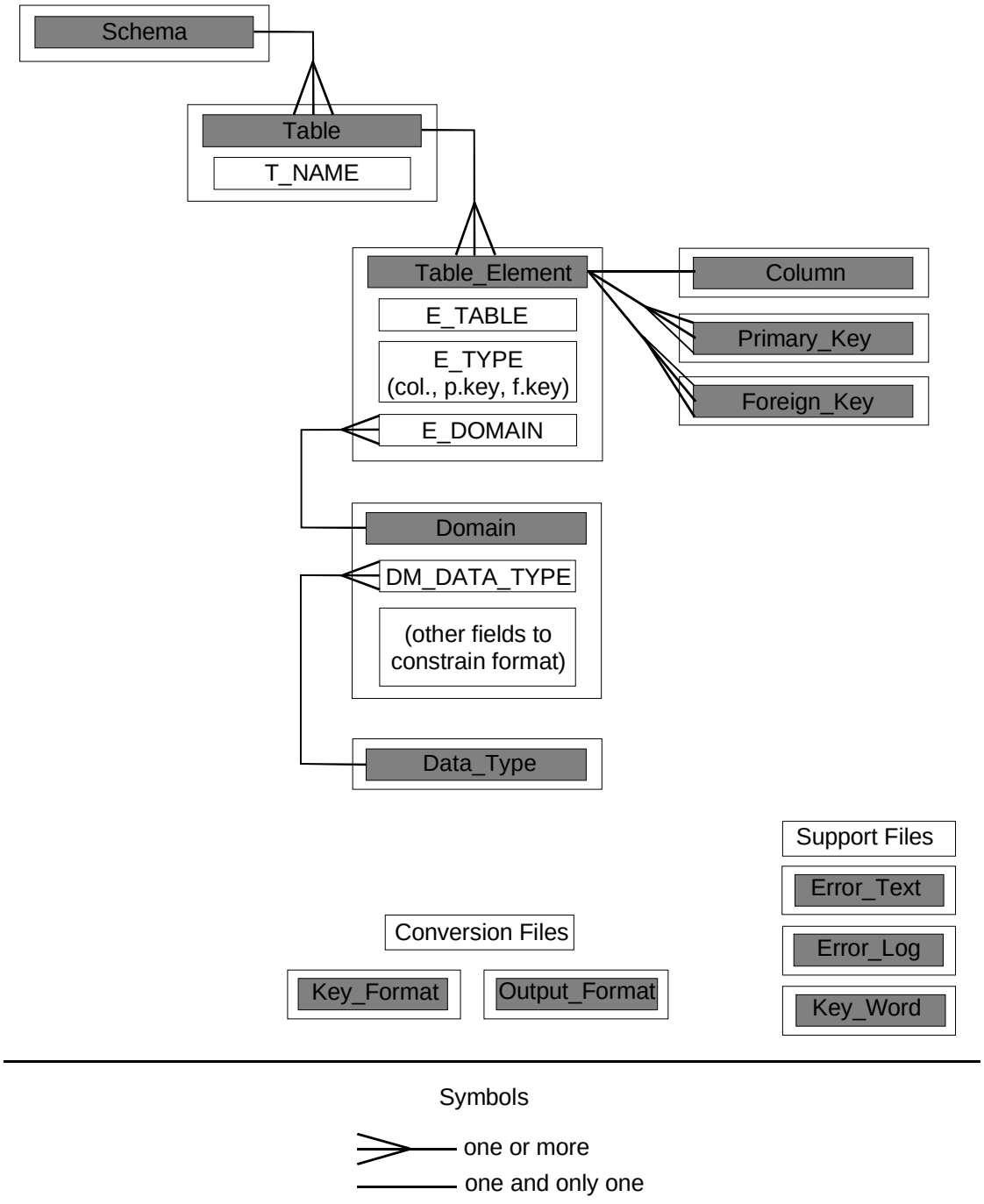
SQLI is implemented as a set of VA FileMan files within a single M global, with no multiples or word processing fields.

The organization of the files mirrors SQL2 standard Data Definition Language (DDL) syntax. Every data structure in the main SQLI files reflects some portion of the DDL commands needed to create SQL data dictionaries for VA FileMan data (essentially, the CREATE TABLE command).

Additional syntax has been added to support the definition of M global structures, virtual columns, key and output formats and other objects outside the scope of the SQL standard.

SQLI File Diagram

This diagram organizes the file entities in their importance to the operation of the SQLI package. It shows conceptual relationships between the files, but not a comprehensive view of the physical pointer relationships between files.



Global Translation, Journaling, and Protection

Translation

Translation is recommended for the sole SQLI global (i.e., ^DMSQ). This global has the potential to be read-intensive as more and more applications are added to it in the future.

Journaling

It is not essential or recommended to journal ^DMSQ. The only **V*IST*A** product using this global is VA FileMan's SQLI.

Protection

The following global protection should be set:

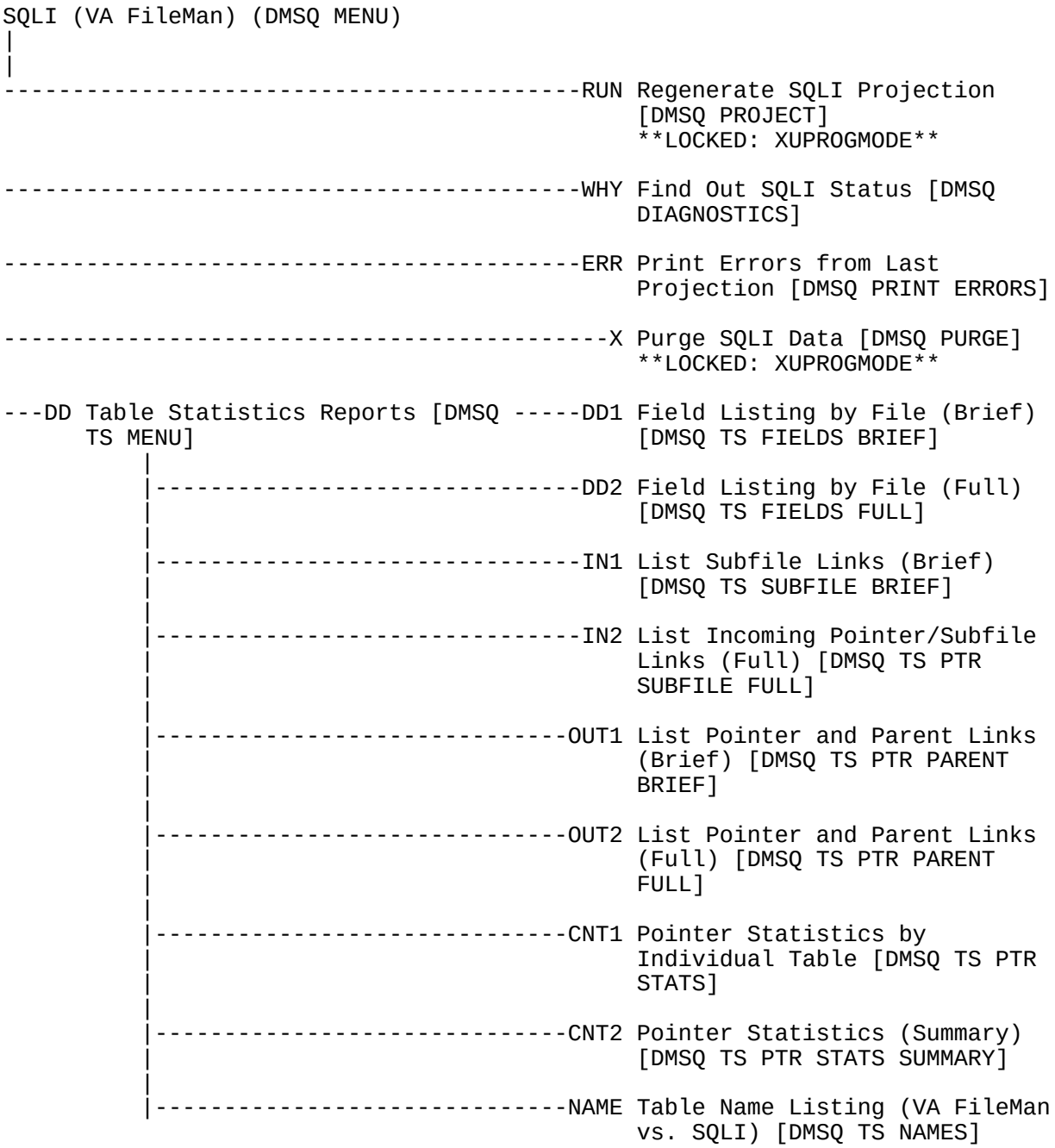
Global Name	Protection		
	DSM for OpenVMS	**OpenM**	**MSM-DOS**
^DMSQ	System: RWD World: RW Group: RW User: RW	Owner: RWD Group N World: N Network: RWD	System: RWD World: RWD Group: RWD User: RWD

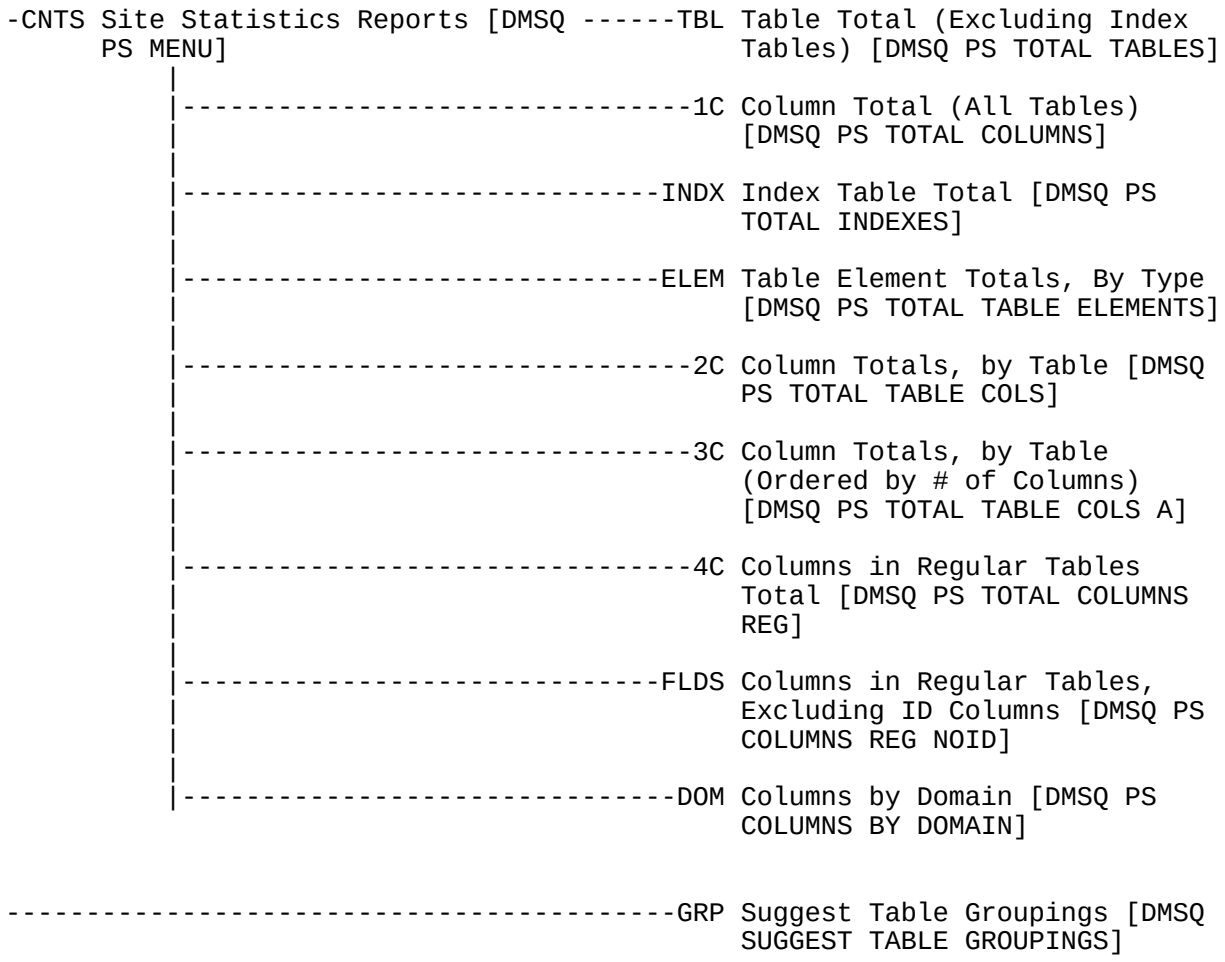
SQLI Routine List

The following routines are distributed in the SQLI DI*21*38 patch:

```
DMSQ, DMSQD, DMSQE, DMSQF, DMSQF1, DMSQF2, DMSQP, DMSQP1,
DMSQP2, DMSQP3, DMSQP4, DMSQP5, DMSQP6, DMSQS, DMSQT, DMSQT1,
DMSQU
```

SQLI Options




-CNTS Site Statistics Reports [DMSQ PS MENU]
TBL Table Total (Excluding Index Tables) [DMSQ PS TOTAL TABLES]
1C Column Total (All Tables) [DMSQ PS TOTAL COLUMNS]
INDX Index Table Total [DMSQ PS TOTAL INDEXES]
ELEM Table Element Totals, By Type [DMSQ PS TOTAL TABLE ELEMENTS]
2C Column Totals, by Table [DMSQ PS TOTAL TABLE COLS]
3C Column Totals, by Table (Ordered by # of Columns) [DMSQ PS TOTAL TABLE COLS A]
4C Columns in Regular Tables Total [DMSQ PS TOTAL COLUMNS REG]
FLDS Columns in Regular Tables, Excluding ID Columns [DMSQ PS COLUMNS REG NOID]
DOM Columns by Domain [DMSQ PS COLUMNS BY DOMAIN]
GRP Suggest Table Groupings [DMSQ SUGGEST TABLE GROUPINGS]

Entry Points

Entry points are supported references that can be called from application code.

SETUP^DMSQ: Generate SQLI Projection (Interactive)

This entry point interactively generates the SQLI projection. It purges the contents of the SQLI files and builds a new projection. Load keywords with the KW^DMSQD prior to calling ALLF^DMSQF.

Format

SETUP^DMSQ

Input

(none)

Output

(none)

ALLF^DMSQF: Generate SQLI Projection (Non-Interactive)

This entry point non-interactively generates the SQLI projection. It purges the contents of the SQLI files and builds a new projection. Load keywords with the KW^DMSQD prior to calling ALLF^DMSQF.

Format

ALLF^DMSQF

Input

(none)

Output

(none)

KW^DMSQD: Add Keywords

For M-to-SQL vendors. Adds reserved keywords to the SQLI_KEY_WORD file (words that shouldn't be used in the naming of files, fields, and other entities). Make sure that a site populates the SQLI_KEY_WORD file with any reserved words **before** the site generates the SQLI projection.

The SQLI_KEY_WORD does not come populated by SQLI. Therefore, M-to-SQL vendors should use the KW^DMSQD entry point to populate the SQLI_KEY_WORD file with:

- The standard set of reserved keywords for SQL as defined by the ANSI standard for SQL
- The keywords for ODBC as defined by Microsoft
- Any keywords specific to your (vendor) M-to-SQL product

Format

KW^DMSQD(global_root [,.output_array])

Input

global_root	(required) Global root of global location containing keywords. For example, if you pass $NA(^TMP("SQA",$J)) as the global reference, then the array of keywords should be in the format:
	
	`^TMP("SQA",132414124,1)=keyword1`
	`^TMP("SQA",132414124,2)=keyword2`
	`(etc.)`

.output_array	(optional) Pass by reference. The array in which error messages should be returned.

Output

output_array	Any error messages that result from trying to add keywords are returned in this array, if the array is passed as a parameter to the call.

Example

```
D KW^DMSQD($NA(^TMP("SQA",$J)))
```

ALLS^DMSQS: Cardinality of All Tables

For M-to-SQL vendors. Calculates T_ROW_COUNT (SQLI_TABLE file) and P_ROW_COUNT (SQLI_PRIMARY_KEY file), for **all** tables projected by SQLI. You can call this after the SQLI projection has been generated.

Format

ALLS^DMSQS

Input

(none)

Output

(none)

STATS^DMSQS: Cardinality of One Table

For M-to-SQL vendors. Calculates T_ROW_COUNT (SQLI_TABLE file) and P_ROW_COUNT (SQLI_PRIMARY_KEY file), for a given table. Call this if you need the cardinality of any table that has **already** been projected by SQLI.

Format

STATS^DMSQS(tableien)

Input

tableien	(required) ien of the table to generate T_ROW_COUNT and P_ROW_COUNT values.

Output

(none)

$$CN^DMSQU

One of SQLI's internal naming functions. Returns a column name unique for a given SQLI table, making sure the name isn't in the SQLI_KEY_WORD file.

Format

```
$$CN^DMSQU(table,column,name)
```

Input

table	Internal record number of table in SQLI_TABLE file.
column	Internal record number of column in question, in SQLI_TABLE_ELEMENT file.
name	Proposed name to use for column (literal string).

Output

Return Value	Unique column name.

Example

```
> W $$CN^DMSQU(297,3407,"NAME")
NAME
```

$$FNB^DMSQU

One of SQLI's internal naming functions. Returns a unique table name compared with existing SQLI_TABLE entries in the same schema, making sure the table name isn't a keyword (in SQLI_KEY_WORD file).

Format

```
$$FNB^DMSQU(file,table)
```

Input

file	VA FileMan file/subfile number.
table	Internal record number of table in SQLI_TABLE file.

Output

Return Value	Unique table name.

Example

```
> W $$FNB^DMSQU(1,580)
FILE1
```

$$SQLI^DMSQU

One of SQLI's internal naming functions. Returns valid SQL identifier based on the input string. Does not check for conflicts with SQLI_KEY_WORD file entries.

Format

```
S X=$$SQLI^DMSQU(string,length)
```

Input

string	Initial string to generate SQL identifier for.
length	Maximum length of generated identifier.

Output

Return Value	Valid SQL identifier based on input string.

Example

```
> W $$SQLI^DMSQU("FILE",30)
FILE
```

$$SQLK^DMSQU

One of SQLI's internal naming functions. Returns valid SQL identifier based on the input string, making sure the identifier isn't a keyword (in SQLI_KEY_WORD file).

Format

```
S X=$$SQLK^DMSQU(string,length)
```

Input

string	Initial string to generate SQL identifier for.
length	Maximum length of generated identifier.

Output

Return Value	Valid SQL identifier based on input string.

Example

```
> W $$SQLK^DMSQU("FILE",30)
FILE1
```

Other Supported References

SQLI Files, Fields and Cross References

All of SQLI's files, fields, and cross-references as distributed in patch DI*21*38 can be referenced directly without integration agreements. This enables M-to-SQL vendors to create SQLI mapping utilities using the SQLI file structures. Specifically, these are the files in the 1.52 to 1.53 number range, all stored in ^DMSQ.

Direct Mode Utilities

Direct mode utilities can be used from programmer mode, but developers may not call them from within applications.

Direct Mode Utility	Description
MAIN^DMSQE	This routine is used by the Print Errors from Last Projection option [DMSQ PRINT ERRORS] to list errors generated during the most recent SQLI projection.
RUNONE^DMSQ	This routine is provided as a troubleshooting utility only. When the SQLI projection aborts due to a data dictionary anomaly, you can use RUNONE^DMSQ to run a "test" projection for a single file or subfile. Primarily this enables you to test whether or not you fixed the data dictionary anomaly, before starting a new SQLI projection for all files on the system.

Appendix A: Case Studies

Case studies are provided in this appendix to give concrete information about how sites are implementing SQLI, what kinds of projects they are trying out, and what kind of experiences they have had using SQLI.

Additional case studies after the publication of this manual may be added to the SQLI web site:

http://www.vista.med.va.gov/softserv/infrastr.uct/sqli/

Profile: San Francisco VA Medical Center

Configuration Information

Site Name:	San Francisco VAMC
Site M implementation:	DSM for OpenVMS AXP
Which database? (production, etc.):	Production
M-to-SQL product used:	KB_SQL V. 3.5
Size of M-to-SQL license chosen (#users):	8

Installation Information

Did IRM install SQLI and M-to-SQL product? Yes
Time to generate full SQLI projection + SQLI mapping: 5 1/2 hours
Disk space used by SQLI projection: 43 megabytes
Disk space used by M-to-SQL mapping: 40 megabytes

User Information

- Purpose of SQL access to VA FileMan data: Web-based telephone directory (for the first SQL project).
- Who are the end users: Web users, including on-site VAMC staff, affiliated medical school users, and VA intranet-wide users.
- Who is DBA? IRM.
- Methods of user access: Web browser.

Project-Specific Details

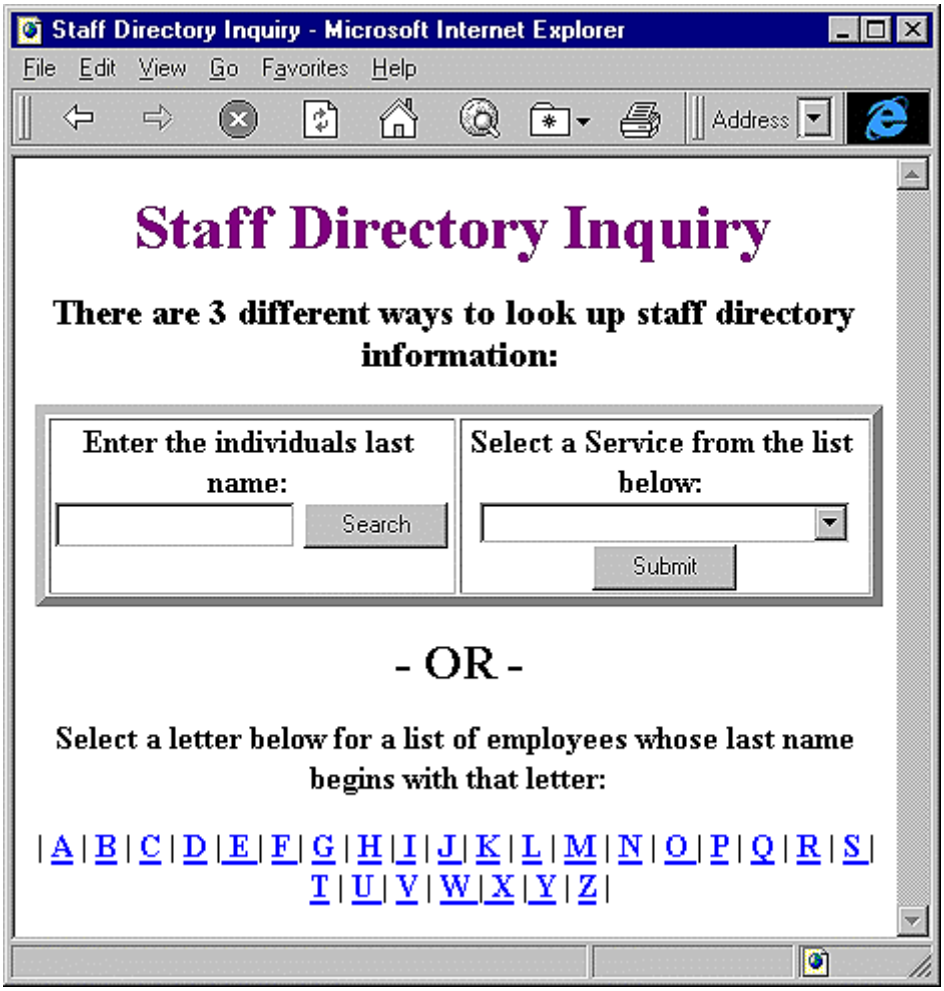
The station's telephone directory is served out on web pages. Several methods of lookup are provided so that you don't have to scan the entire directory list.

The telephone directory application is based on the use of Active Server Pages (ASPs). ASPs use server-side scripting to retrieve records from a database and return them as table rows in pre-formatted web pages.

Microsoft's Visual InterDev was used to build the server-side scripting in the ASPs. Each embedded script that executes on the server uses ODBC to perform a particular database query.

The web server (a Pentium running Windows NT) is set up with an ODBC driver that connects over TCP/IP to KB_SQL running on the production M system.

KB_SQL acts as an ODBC data source, with an ODBC listener process running continuously. KB_SQL's data dictionaries are mapped based on SQLI to directly access VA FileMan data as tables.



Pages that Retrieve a List of VA FileMan Records

For a web page that is going to list multiple records, a table is used, with one record listed per table row. Using Visual InterDev, a data range header and data range footer control is inserted on the web page. The data range header performs a "select" query (or can reference a stored procedure, which executes a "select" query on the server).

Between the controls, HTML code is used that creates a single row within a table (but not the table itself). A retrieved column value can be inserted in the row's table cells using the ASP syntax <%= DataRangeHdr1("column_name") %>. When the page is displayed, one table row is displayed per retrieved record.

Pages that Retrieve and Display a Single VA FileMan Record

For a web page that retrieves and displays a single record, you can present the data from the record in any format you choose (not just in a table). Using Visual InterDev, a data command control is inserted on the web page. This control contains a query that should retrieve a single table row. Then, in the HTML elsewhere on the web page, you can insert values from the record using the ASP syntax <% DataCommand1("column_name") %>.

Hurdles

- Variable Pointer Fields

  When accessing a variable pointer field through ODBC, only the unresolved internal value of the pointer field was returned. To get the resolved external value of a variable pointer field, a new free text field was made for the file in question. A trigger cross-reference was added to the variable pointer field. The trigger obtains the resolved variable pointer field value and stuffs it into the free text field. This free text field can then be retrieved through KB_SQL.

- Foreign Keys - How to Use through ODBC

  KB_SQL supports the use of foreign key syntax, which allows the values of fields in tables that have a "join" relationship to be accessed without having to do an actual join between the two tables. The main advantage of KB_SQL's foreign key (implicit join) syntax over joins is that rows are not lost when the values of join columns are blank. For example, if you are joining two tables based on the column for a pointer field, some entries may have a null value for the pointer field - but that table row was still desired.

  KB_SQL's foreign key syntax works correctly when a query is done entirely in KB_SQL. When query results are returned through ODBC, however, there is an error because ODBC doesn't recognize columns retrieved through resolving foreign keys. The solution was to define an alias for the resolved foreign key (e.g., SERVICE_SECTION_FK@NAME AS SERV_NAME). The column values are returned through ODBC under the alias SERV_NAME.

- Stored Procedures - Increased Performance

  KB_SQL stored procedure provide a performance advantage over executing an SQL query passed whole through ODBC to KB_SQL. This is because stored procedures call a query that is already compiled in KB_SQL, whereas a query issued directly is compiled by KB_SQL each time prior to being executed.

Profile: Brooklyn VA Medical Center

Configuration Information:

Site Name:	Brooklyn VAMC/VISN 3
Site M implementation:	DSM
Which database? (Production, etc.):	Test now; Production eventually
M-to-SQL product used:	KB_SQL
Size of M-to-SQL license chosen (#users):	8
Size of Site Database:	1 gigabyte (Test)

Installation Information

Did IRM install SQLI and M-to-SQL product? No; KB Systems did
Number of tables projected: 4000
Length of running SQLI mapper, and SQLI mapper: 3.25 hours (both together)
Size of SQLI files after projection: 25 megabytes
Disk space used by M-to-SQL mapping: 25 megabytes

User Information

- Purpose of SQL access to VA FileMan data: Let users export data to their PCs so they can create their own custom reports.
- Who are the end users: IRM and selected end-users.
- Who is DBA? An IRM developer.
- Methods of user access: Through the MedRecord Gateway for Windows product.

Project-Specific Details

The goal of this project is to enable end-users to create their own custom reports on VA FileMan data. Without this project, end-users typically have IRM create custom VA FileMan reports for them. IRM will also generate some of its own reports using the tools provided by this project, as well as creating traditional VA FileMan reports.

The product used as the front end to this project is MedRecord Gateway for Windows (MGW) from Strategic Reporting Systems, Inc. It runs as an ActiveX control in the Internet Explorer web browser.

The MGW part of the solution is analogous to a GUI client version of the VA FileMan Export tool, using a forms-driven SQL-based interface. The View Manager module allows privileged users to create views for end-users. The Wizard module allows web users to authenticate themselves, create a query against tables or a pre-defined view in a GUI interface, retrieve the corresponding table rows and load them directly into Excel (currently) and ReportSmith (future).

The total solution, wherein users can enter queries and get the results in Excel or ReportSmith, amounts to an alternative report generator module for VA FileMan.

View Manager Module

MGW's View Manager module allows IRM and other users with sufficient privileges to define views for use by other end-users. It is implemented as an ActiveX control that is used through a web browser.

Wizard Module

MGW's Wizard client module is analogous to VA FileMan's Export tool, with a number of advantages including a forms-driven GUI interface for query creation and seamless retrieval of data from host to client. It can be used as a client to access a VA FileMan database, a Microsoft SQL Server database, and a Microsoft Access database. The Wizard module is implemented as an ActiveX control that is used through a web browser.

Connection to the VA FileMan database (through KB_SQL and SQLI) is handled with the KB_SQL ODBC driver, which is installed on each client. The ActiveX control uses the ODBC driver transparently to retrieve data from the server.

Data is retrieved using the following six basic steps:

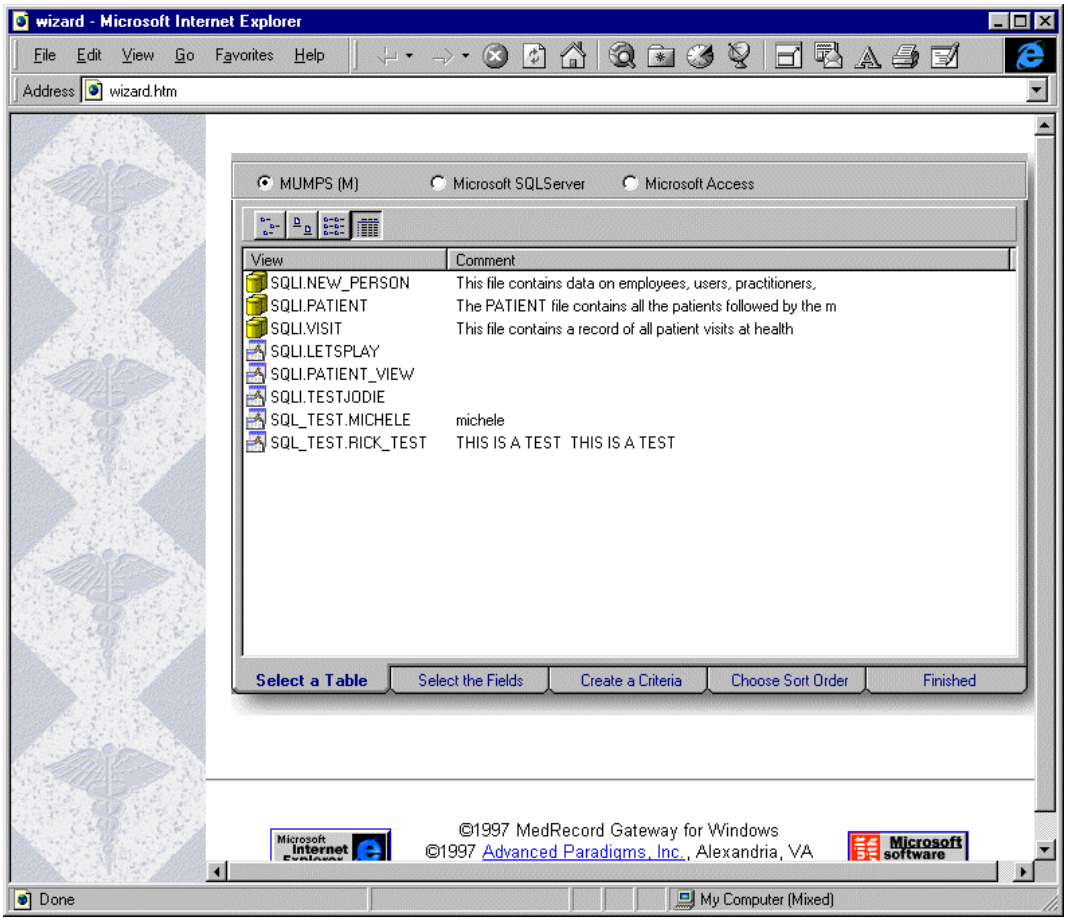
1. Select tables or view for the query.
2. Select the fields to use in the query.
3. Set query criteria.
4. Choose the sort order for the returned table rows.
5. Execute query (rows are returned to the Wizard ActiveX control.)
6. Export data from the Wizard control to Microsoft Excel.

Screen captures* of a simple query on the NEW PERSON file using the MGW Wizard module are provided on the following pages.

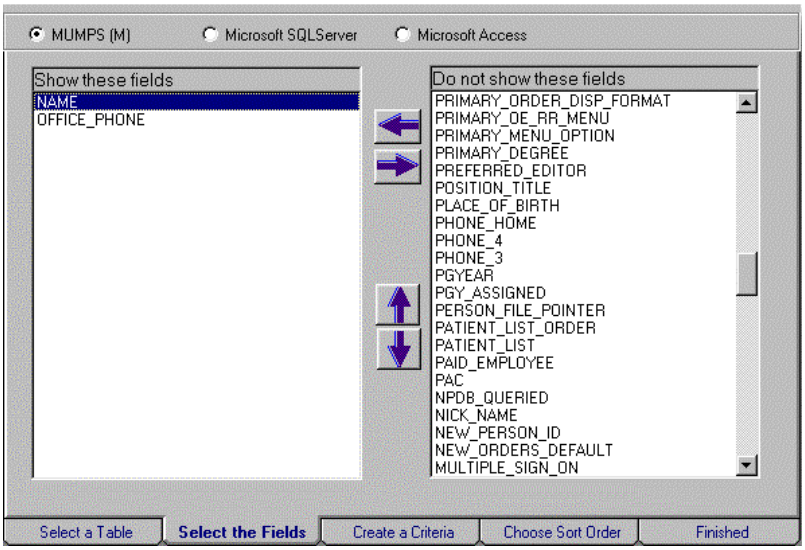
* Images used by permission of Strategic Reporting Systems, Inc.

Steps 1 and 2: Select Tables or View, and Select Fields

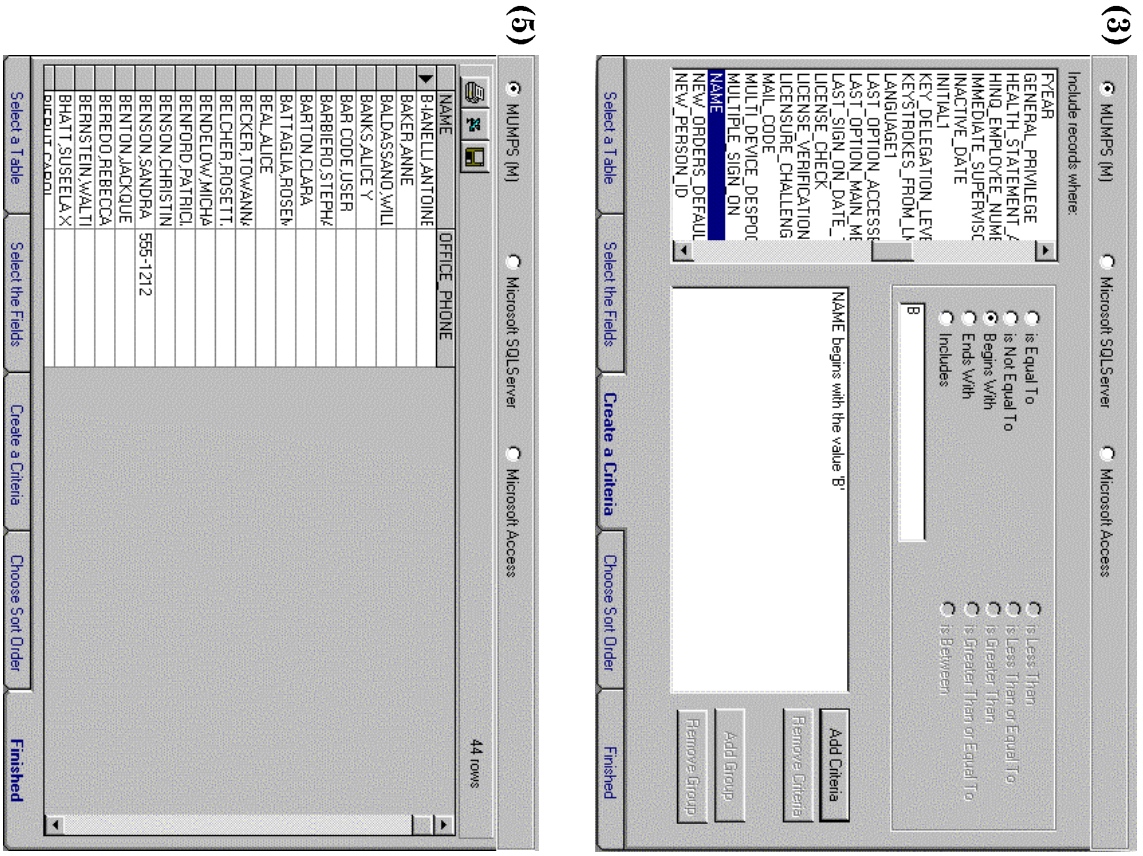
(1)



(2)



Steps 3-6: Choose Criteria and Sort Order, Execute Query, Export to Excel



Glossary

Base Value - The stored value of a column in SQL, not transformed in any way.

Cardinality - The cardinality of a table is its number of rows; the cardinality of a domain is the number of possible values in the domain.

Column - A set of values for a particular value sequence in a row, for each row in a table (akin to a VA FileMan field). All values in a column must be of the same data type or domain.

Data Type - A set of possible values. SQL has its own set of standard data types; SQL vendors often implement additional data types.

Data Dictionary - is a file that defines a file's structure, to include a file's fields and relationships to other files.

DBA - Database Administrator for an SQL system. The DBA has, by default, full privileges to every object in the database.

DBS - Database Server. DBS is a non-interactive VA FileMan API. It makes no writes to the screen. It provides client/server access to VA FileMan data. DBS calls of particular interest to M-to-SQL vendors using SQLI include $$GET1^DIQ, FIELD^DID, and $$EXTERNAL^DILFD.

DCL - Data Control Language. The set of SQL statements through which access to the database is controlled.

DDL - Data Definition Language. The set of SQL statements through which objects are created and modified in the database.

DML - Data Manipulation Language. The set of SQL statements through which data is modified.

Domain - A set of permissible values. A domain is based on a data type, but may contain further constraints on what values are valid for the domain.

Extract Storage - When the storage location for a particular VA FileMan field is designated to be by position on a global node, instead of being character-delimited.

Field Type - The type of VA FileMan field. There are nine FileMan field types. VA FileMan field types loosely correspond to the concept of *data type.*

Foreign Key - A foreign key acts as a ready-to-use join between two tables. It matches a set of columns in one table to the primary key in another table.

Hierarchical Database - A database structure in which files can own or belong to each other. Often referred to as a parent-child structure.

Ien - Internal entry number. This is the numeric subscript beneath a file's global root under which all of the data for a given VA FileMan file entry is stored.

Ien Column - A column SQLI projects to contain the ien of a VA FileMan entry.

Join - In SQL, a join is when two or more tables are combined into a single table based on column values in an SQL SELECT statement.

M-to-SQL Product - Software that can view structured M globals as relational tables through SQL.

Multiple-Valued Field - A VA FileMan filed that allows more than one value for a single entry. See also Subfile.

ODBC - Open Database Connectivity. ODBC is Microsoft's solution to enable client access to heterogeneous databases.

Outer Join - A join between two tables, where rows from one table are present in the joined table, even when there are no corresponding rows from the other table.

Output Formats - Output formats are provided by SQLI to convert column base values into a format suitable for external use by end-users.

Primary Key - a designated set of columns in a table whose values uniquely identify any row in the table.

Query - An SQL command that extracts information from an SQL database.

Relational Database - A database that is a collection of tables, and whose operations follow the relational model.

Row - A sequence of values in a table, representing one logical record.

Schema - A schema defines a portion of an SQL database as being owned by a particular user.

SQL - Structured Query Language, the predominant language and set of facilities for working with relational data. The current ANSI (American National Standards Institute) standard for SQL is X3.135-1992.

SQLI Mapper - Software written by an M-to-SQL vendor that maps the vendor's SQL data dictionaries directly to VA FileMan data, using the information projected by SQLI.

Subfile - The data structure of a multiple-valued field. In many respects, a subfile has the same characteristics as a file.

Table - A collection of rows, where each row is the equivalent of a record. A base table (one not derived from another table) is the SQL equivalent of a database file.

Table Element - a column, primary key, or foreign key that is part of a table.

View - A user-defined subset of tables, based on a SELECT statement, containing only selected rows and columns.

.01 Field - A field that exists for every VA FileMan file, and that is used as the primary lookup value for a record.

Index